Traffic Forecasting in Smart Cities

Final Master Thesis

Juan José Vázquez Giménez

Advisor	Mari Paz Linares Herreros - inLab FIB
Co-advisor	Jamie Arjona Martínez - inLab FIB
Tutor	Josep Casanovas Garcia
Tutor department	Department of Statistics and Operations Research (DEIO)
Master	Master in Innovation and Research in Informatics
Specialty	Data Science
Date	15 de gener de 2019

Abstract

Nowadays, cities must address the challenge of sustainable mobility. Traffic state forecasting plays a key role in mitigating traffic congestion in urban areas. For example, predicting path travel time is a crucial issue in navigation and route planning applications. Furthermore, the increasing penetration of Information and Communication Technologies makes Floating Car Data become an important source of real-time data for Intelligent Transportation Systems applications.

In the context of an educational cooperation agreement with the research group of the inLab FIB focused on smart mobility, this master thesis deals with the problem of urban traffic forecasting when Floating Car Data is available. The main goal is to perform traffic forecasting with machine learning methods and evaluate this kind of solutions under different conditions: size of the network, penetration rate of the floating cars, prediction horizon and the amount of required data.

The current state of the art shows that most of the newly proposed methods for urban traffic forecasting use a Deep Learning approach. A comparison of four neural network approaches (recurrent and convolutional) is presented to evaluate the capabilities of Deep Learning methods to solve the problem.

Different tests are proposed in order to evaluate the developed Deep Learning models, and also to analyze how the different proposed factors affect the forecasting accuracy. The performed experiments are designed through a microscopic traffic simulation approach to emulate Floating Car Data.

The main conclusions from the obtained results show that the methods without a convolutional component present the best performance for all the tested experimental scenarios.

Keywords: urban traffic forecast; deep learning; floating car data.

Acknowledgements

Ahora que estoy en el final de este camino echo la vista atrás y me doy cuenta de la cantidad de gente que ha aportado su granito de arena a este proyecto. Personas que en mayor o menor medida me han ayudado a llegar hasta aquí, a veces (incluso) sin saberlo. Me gustaría agradecer a todos aquellos que, aunque sus nombres no aparezcan entre estas líneas, han sido muy importantes para mi durante esta etapa. Dicho esto, considero que existen ciertas personas sin las cuales no habría sido posible y a los que quiero mencionar de forma especial.

En primer lugar, a Mari Paz por depositar su confianza en mi. Desde que surgió la idea por primera vez, hasta la realización de esta, ha luchado a mí lado por este proyecto, incluso más de lo que requiere su papel de directora. En todo momento he contado con su apoyo, tanto a nivel profesional como personal, ayudándome siempre que lo he necesitado. También a Jamie, porque es un privilegio contar con un co-director como él. El hecho de que este proyecto tenga un enfoque similar a su tesis, aun tratando temas diferentes, le convierte en alguien idoneo con el que compartir problemas, dudas y retos comunes. Además, su admirable pasión por formarse así como su inquietud por aprender algo nuevo cada día le hacen el compañero de aventura perfecto. Es una suerte que personas como ellos sean mis directores, pero sobretodo mis amigos.

A Josep, por sus consejos, por darme la oportunidad de pertenecer a inLab y por hacer posible este proyecto. Me parece una labor excepcional la que se hace desde el laboratorio, ayudando a los estudiantes a formarse a través de su participación en proyectos reales y apostando por la transferencia de conocimiento desde la universidad.

Al increíble equipo del inLab, tanto estudiantes como personal, por convertir el entorno de trabajo en un lugar a donde te apetece ir cada mañana. En especial a Marta, por cuidar de todos y no tirar nunca la toalla. A Gonzalo, Albert, Pau, Sergio y Tote por intentar entender lo que escribo y ayudarme a expresarlo mejor. Y a Juan y Josefran, por cierta tarde de horas interminables viendo como la temperatura de la GPU subía y bajaba.

Y por último, y no por ello menos importante, a mi familia y amigos por preocuparse de mi en los peores momentos y ofrecerme una válvula de escape cuando más lo necesito. En especial a Sara, que ha sufrido en primera persona todos los altibajos de este proyecto con una paciencia inagotable, sacándome siempre una sonrisa. ¡No podemos formar mejor equipo!

¡Gracias a todos!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This initial Chapter introduces the context of the proposed master thesis carried out for the *inLab FIB*[1], the innovation and research laboratory of the Faculty of Informatics at Universitat Politècnica de Catalunya. In addition, the traffic forecasting problem is presented, including a description of its key issues. Furthermore, main objectives and motivations of the project are exposed, together with some research questions. In addition, the temporal planning of the whole project is described. Finally, an outline of the whole document is presented in order to expose the applied methodology and organize its content.

1.1 Context

According to the data published by the European Comision in 2017[2], urban areas are the "engine" of economic growth and employment, as around 85% of the European Union's gross domestic product is generated in cities. Therefore, the population is more and more concentrated in the cities. In 2010, 73% of European citizens lived in urban areas and this percentage is expected to increase to over 80% by 2050. Due to the population concentration growth, cities face multiple problems related to, or caused by, transport and traffic. One of the most important problems is the private vehicle use increment and with it, the urban networks saturation.

Urban traffic saturation is a problem in multiple ways. Figure 1.1 shows the result of the article named *The hidden cost of congestion*[3] published by *The Economist* publication in 2018. On the left is shows the hours drivers spent of some cities in peak traffic congestion during 2017. Time wasting implies a money wasting too, the economic cost per driver are quantified in the right figure (in thousands of dollars). Also, the economic costs (in billions of dollars) that it means for the cities of United States, United Kingdom,

[1] `https://inlab.fib.upc.edu/`
[2] `https://ec.europa.eu/transport/themes/urban/urban_mobility_en`
[3] `https://www.economist.com/graphic-detail/2018/02/28/the-hidden-cost-of-congestion`

and Germany. But urban traffic has not only time and economic effects according to the *World Health Organization* the exposure to ambient (outdoor) air pollution causes 4.2 million deaths every year. The same organization classifies the fuel combustion from motor vehicles (e.g. cars and heavy-duty vehicles) as one of the major human activities sources of outdoor air pollution. Accidents on the roads and noise pollution are also bad consequences of the traffic, especially in urban areas.

All these problems can be relieved through the decrease of road traffic congestion, especially in urban cores. Lana et al. [2018] classify the different strategies of mitigation of traffic congestion in three: increasing and improving infrastructures, proposing new transport alternatives and managing traffic flows. Due to the difficulties presented by the first one (limited by topographical, budgetary and social factors) and the dependency of the second one to political decisions, the improvement of the traffic management is the main followed strategy. In the last years, the development of new technological solutions, the computational capability of systems and the quantity of data generated have been incremented. These factors favor the development and the use of new Traffic Management Systems (TMS) and Traveler Information Systems (TIS). Some of the key features for this kind of systems need traffic forecasting, for example to anticipate actions for future situations or to perform navigation and route planning taking into account the future traffic state. The following Section describes the traffic forecasting problem and what aspects are used to differentiate between its subproblems.

1.2 The traffic forecasting problem

The traffic forecasting problem is very extensive and it includes some subproblems according to different issues. One of the most important features in the traffic forecasting problem is the kind of network where the predictions are performed. Usually, they are classified in urban networks and freeways. The topology is very different because urban networks contain more and shorter links while freeway networks are composed of few but larger links. Also, some authors include the arterial network category when the prediction is performed only over the main sections of an urban network. Typically, forecasting in urban areas is more difficult because the behavior of the drivers inside the network is less predictable.

Another key feature in this problem is the prediction horizon. The literature uses the expressions short-term and long-term to classify them. Although it is not clear exactly what horizons each group refers to, the short-term name is used for predictions from 1 minute to around 30 minutes or 1 hour depending on the author. For larger prediction horizons, the long-term name is used. Of course, despite this binary classification, the prediction horizon can variate between 1 minute to hours or days. In general, the complexity of the problem increases and the forecasting accuracy decreases with larger horizons.

Besides the kind of network and the horizon time, the traffic prediction problem is determined by what variable and with which granularity is predicted. Traffic forecasting can be performed for different variables, the most presents in the literature are four:

- Traffic flow: it is the number of vehicles that pass through some site in a determined amount of

Figure 1.1: Results of the article named *The hidden cost of congestion* published in 2018. *Source:* `https://www.economist.com/`

time (measured in vehicles/second or vehicles/hour).

- Traffic density: it is the number of vehicles located in a determined area at the same time (measured in vehicles/meter or vehicles/kilometer).
- Average speed: it is the average speed for the vehicles in a site (measured in kilometer/hour or meter/second).
- Travel time: it is the time that takes a vehicle to travel from a origin point to its destination (measured in seconds, minutes or hours).

These predicted variables can be used in different scales, like a specific point in a network, a section (a link or a part of a link) of the network and the whole network (or a sector for the network).

Also, as seen on most of the machine learning problems, the prediction performed can be regression or classification. This depends on if the prediction is performed over a continuous number (the presented predicted variables) or over a finite set of values (a discretization of the previously predicted variables). So, depending on the final goal, the traffic forecasting solution could try to advance the average speed of each section, the general traffic state of a network (for example free, medium or congestion), the number of vehicles that will pass through some point, etc.

Finally, the last issue to be considered is the data source used. Nowadays, traffic data is generated in multiple ways and many types of systems can be used to it. Following, the most usual traffic data sources are listed:

- User surveys: Traditionally, most of the other options did not exist and the traffic was measured through surveys performed directly over the population. This data source has been replaced by other automatic ways (mentioned below) which are cheaper and collect better data.
- Sensors: These devices are able to register some traffic features like the presence of a vehicle, its speed, etc. In the last years and following with the smart cities evolution, the presence of traffic sensors has grown and they have been improved. For example, one of the first used sensors was the loop detector which is able to detect vehicles in a specific position. This kind of devices requires roadworks for their installation, thus their relocation is too expensive. Currently, other modern options like the ANPR (Automatic Number-Plate Recorder) are used, with an easier installation and the capability to identify which cars pass through a point. Despite these improvements, they are only able to register the activity in a fixed location and the quantity of them needed to cover a whole city is very high.
- Cameras: Although the main goal of traffic cameras is to offer a real-time monitorization of the network state, in recent years they have been used as traffic data collectors. It is a very good way to reuse installed systems, but the use of camera records as traffic data needs a computer vision process to translate the images into, for example, traffic flow data. In addition, they present the same problem than the sensors about the fixed position.

- GPS-FCD: The systems installed in the modern vehicles allows to locate the vehicles in real time with high precision. This kind of data is the most desirable for the traffic forecasting systems that offer individual data of high quality and without location limitations. The main problem is that a sufficient penetration rate is needed in order to this data being representative.
- Cellphones: Because nowadays most people bring connected smartphones with them (even while driving), the data generated by these devices can be used as GPS-FCD.

In addition to traffic data, the use of some external data is more and more usual in the literature. This information is named exogenous variables and its use allows to adjust the predictions to some external conditions. For example, in the temporal dimension, the use of information like the moment of the day, the day of the week, the season of the year or the holiday days can be decisive to improve the forecasting accuracy. Also, other factors that can change the traffic situation are the weather conditions, the city events (special and periodical), the roadworks, the traffic incidents, etc. These ones are also used as exogenous variables in the literature.

1.3 Objectives and motivations

The main objective of this master thesis is to perform traffic forecasting in urban contexts by developing machine learning models trained with floating car data (FCD). Concretely, the average speed is predicted for each road section of the whole urban scenario. Thus, regression machine learning methods must be used because the answer variable is numeric. In addition, this work is not exclusively interested in only short-term or only long-term forecasting so both approaches are tested.

This project is focused on smart cities scenarios where FCD is available. Currently, gathering real data in order to evaluate different FCD penetration levels is a utopia. Thus, these scenarios are emulated with traffic simulation. The possibility of analyzing how machine learning approaches perform in a smart city and what aspects affect the accuracy of this kind of solutions can help in the design and decision making during a smart city project development. Also, the evaluation of the minimal requirements for a traffic forecasting model is very useful to study their real feasibility.

Furthermore, another motivation exists for the research proposed in this master thesis, which is developed for the inLab FIB smart mobility research group. Traditionally, the group solves smart city problems performing analysis and forecasting of different traffic situations through simulation techniques. Development and calibration of a simulation model are costly tasks that require access to real data, which sometimes is difficult to obtain. This is critical in order to adjust the model to the real scenario. On the other hand, due to the increment of the quantity and the different sources of data, some problems can be faced from a data-driven approach. This kind of solutions, like the machine learning methods, does not need the development of a simulation model. So, with enough data, a data-driven approach can be cheaper than a simulation one in terms of time and money. This project will try to solve the traffic forecasting problem with machine learning methods as an alternative to the simulation methods.

In addition to the main objective, we are interested in the following research questions:

- How does the penetration rate of floating cars in the city affect the traffic forecasting performance? Which is the minimal penetration rate to achieve an accurate traffic prediction?
- Which of the tested machine learning methods performs more precise predictions? Does it depend on the traffic context or there is always a better method?
- In traffic simulation approaches, a network model of the simulated area must be developed. Could this topological information be used to improve the results of traffic forecasting? How useful is it?
- How does the size of the urban network affects the traffic forecasting accuracy and the required training time? Is it feasible to perform traffic predictions in big urban networks?
- How does the prediction time horizon affect the accuracy of the prediction?
- How much data is needed to train the machine learning traffic forecasting models?

1.4 Temporal planning

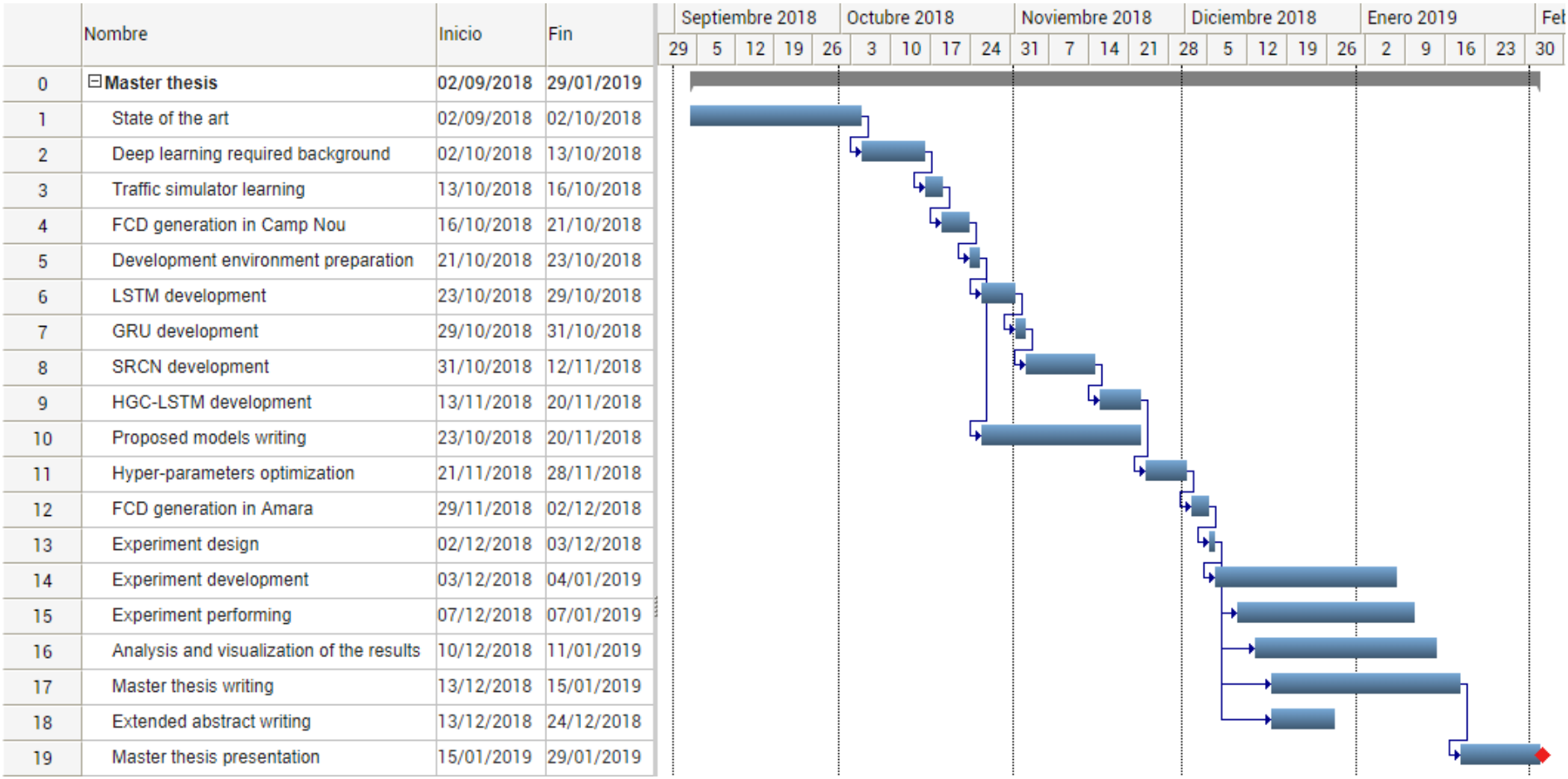
According to the current academic regulations for the Master thesis of MIRI[4], approved in March 19th 2014, this master thesis corresponds to 30 ECTS. The workload estimated for each ECTS is of 30 hours. Thus, the workload of this master thesis must be of $30 * 30 = 900$ hours.

This is a B modality master thesis project and it has been developed under an educational cooperation agreement with the smart mobility department of the inLab FIB. The main tasks that compose this master thesis are described below and their time planning is scheduled in Figure 1.2.

- *State of the art:* The first task of the project is the literature review and the selection of the proposals that will be developed. This task involves the writing of Chapter 2.
- *Deep learning required background:* Once the methods to develop are selected, the study of the required background is needed. This task involves the writing of Chapter 3.
- *Traffic simulator learning:* The generation of the data using the traffic simulator requires a previous task to learn to use the tool.
- *FCD generation in Camp Nou:* Once a basic knowledge of the traffic simulator is achieved, the data generation strategy can be defined and the first scenario can be used to generate the needed data. This includes implementing the required simulator scripts.
- *Development environment preparation:* Before of starting with the methods development, the most adequate tools are selected and the needed environment has to be installed and configured.

[4] `https://www.fib.upc.edu/en/studies/masters/master-innovation-and-research-informatics/final-master-thesis`

- *LSTM development:* When the floating car data is available, the first method can be developed including all the previous data processing.
- *GRU development:* With the LSTM method developed, the easiest method to implement is GRU method, which is similar to the previous one.
- *SRCN development:* Following with the methods development, the third one implemented is the SRCN. It requires to perform a new data processing and incorporate the convolutional components. Also, new data is required from the simulation scenario and this requires the implementation of a new simulator script.
- *HGC-LSTM development:* Lastly, the HGC-LSTM proposal is implemented. The structure of the method is the same than in the SRCN case. But, this method presents a new input representation and it has to be developed. Also, new data is required from the simulation scenario and this requires the implementation of a new simulator script.
- *Proposed models writing:* During the previous four tasks, the writing of Section 4.1 is involved.
- *Hyper-parameters optimization:* Once the four methods are implemented, the hyper-parameter optimization strategy has to be selected and implemented. This task involves the writing of Subsection 5.3.
- *FCD generation in Amara:* When all the methods are developed and ready to be used, the data for the second network is generated.
- *Experiment design:* With all the methods developed and the data generated, the project objectives are revisited and the experiments are designed in consequence.
- *Experiment development:* In function of the designed experiments, the needed testing framework is developed to perform them.
- *Experiment performing:* Lastly, the experiments are executed in order to collect the desired results.
- *Analysis and visualization of the results:* With the first experiments performed, the analysis and the visualization of the obtained results are implemented in order to include them in the report.
- *Master thesis writing:* Although some report parts are already written, at the end of the project development some content has to be redacted.
- *Extended abstract writing:* Part of the work of this master thesis is included in an extended abstract and submitted to the *22nd Meeting of the Euro Working Group on Transportation (EWGT2019)*. The document, named *A Comparison of Deep Learning Methods for Urban Traffic Forecasting using Floating Car Data. J.J.Vázquez, J. Arjona, M.P. Linares, J. Casanovas-Garcia*, is wrote also during the project development.
- *Master thesis presentation:* Finally, the presentation of this master thesis is prepared.



	Nombre	Inicio	Fin
0	**Master thesis**	02/09/2018	29/01/2019
1	State of the art	02/09/2018	02/10/2018
2	Deep learning required background	02/10/2018	13/10/2018
3	Traffic simulator learning	13/10/2018	16/10/2018
4	FCD generation in Camp Nou	16/10/2018	21/10/2018
5	Development environment preparation	21/10/2018	23/10/2018
6	LSTM development	23/10/2018	29/10/2018
7	GRU development	29/10/2018	31/10/2018
8	SRCN development	31/10/2018	12/11/2018
9	HGC-LSTM development	13/11/2018	20/11/2018
10	Proposed models writing	23/10/2018	20/11/2018
11	Hyper-parameters optimization	21/11/2018	28/11/2018
12	FCD generation in Amara	29/11/2018	02/12/2018
13	Experiment design	02/12/2018	03/12/2018
14	Experiment development	03/12/2018	04/01/2019
15	Experiment performing	07/12/2018	07/01/2019
16	Analysis and visualization of the results	10/12/2018	11/01/2019
17	Master thesis writing	13/12/2018	15/01/2019
18	Extended abstract writing	13/12/2018	24/12/2018
19	Master thesis presentation	15/01/2019	29/01/2019

Figure 1.2: Gantt diagram of the main project tasks planning.

1.5 Master thesis outline

Considering the above-mentioned objectives, the work performed in the present project is organized as follows:

- **Chapter 2** makes a review of the current state of the art, as well as the proposed methods made in the literature with significant contents for this research. In addition, taking into account the objectives of the project, the most suitable methods to be implemented are selected.
- **Chapter 3** describes the neural network concepts required to understand ths development details of the previously selected methods.
- **Chapter 4** develops the models to perform urban traffic forecasting based on the selected proposals from the state of the art. Also, it includes the details about their implementation.
- **Chapter 5** presents the performed computational experiments to analyze and evaluate the developed solutions in different urban scenarios. Furthermore, the data generation process using microscopic traffic simulation is also summarized. Also, this Chapter includes the hyper-parameter optimization for the forecasting models.
- **Chapter 6** discusses the contributions of this master thesis and further research.

Chapter 2

State of the art

This Chapter presents the state of the art in traffic forecasting. The traffic forecasting topic has been an active research area since the late 1970s and the existing literature is voluminous. This is due to the importance of the modern Intelligent Transportation Systems, including both Advanced Traffic Management Systems and Advanced Traveller Information Systems, and the fundamental role of the traffic forecasting in these systems. As it is shown in Section 1.2, the traffic forecasting problem is too general and it includes a lot of different sub-problems, of different complexity degrees depending on some mentioned aspects. Because of this, some survey papers such as Vlahogianni et al. [2014] and Lana et al. [2018] exist in order to arrange the existing literature, analyze the evolution, the new trends and focus the research on unsolved problems of the area.

In Lint and Hinsbergen [2012] a classification of models used for traffic forecasting is presented. They can be classified in **Naive**, **Parametric** and **Non-Parametric** methods.

- **Naive methods:** The Naive methods are the simplest methods and do not make any model assumption. The typical Naive methods are the Instantaneous Travel Time, which supposes that current values remain constant indefinitely, and Historical Averages, which consider that the predicted value is the average of previous values under some filters such as day of the week or hour of the day.
- **Parametric methods:** The structure of the Parametric models is predetermined on the basis of theoretical considerations where the parameters are fitted using data. This kind of methods is based on traffic flow theory and simulation on different levels: Analytical approaches and macroscopic, mesoscopic and microscopic traffic flow models.
- **Non-Parametric methods:** The Non-Parametric methods refer to the models where its structure and the values of its parameters are determined from data.

In Figure 2.1, a diagram of the traffic forecasting methods classification is included in order to visually

arrange them. The clear boxes contain the methods more studied in this state of the art.

Since the first researches in this topic, the employed approaches have evolved from **Naive** and **Parametric** methods to **Non-Parametric** methods. Inside the **Non-Parametric** methods used, one of the most common ones in the literature, especially at the beginning, are the time series. They are applied to data with a temporal structure and consist of modeling a response variable using its past observations (autoregressive) and past errors (moving average). The Autoregressive Moving Average models assume that the process generating the data is stationary, but in the traffic case, this is not true. The Autoregressive Integrated Moving Average (ARIMA) models include this consideration, adding the term "integrated" (referent to structural trends in the data). Some different versions of the ARIMA models are described in the literature such as: including periodic terms such as the hour of the day (Seasonal ARIMA model), changing a single response variable by a vector (Vector ARIMA model) or adding spatiotemporal relations (Spatio-Temporal ARIMA model). Although few new studies propose these methods, they are still used as comparative models.

Due to the increment of the quantity and different sources of data, together with the computational capability of the new systems, the trend of the last years has changed in favor of the **Non-Parametric** methods, concretely machine learning methods. So, in following Sections of this state of the art are focused on this kind of methods. In order to arrange the revised literature, the papers are classified by the kind of machine learning methods that they use. Aside from that, given the big quantity of studies in the traffic forecasting and the main objectives of this work mentioned in Section 1.3, the research is focused in publications that perform traffic forecasting in Urban contexts and/or uses Floating Car Data (FCD) as the data source to perform the predictions. The following sections include papers mostly located between 2004 and 2018 classified by the most common methods found in the literature: k-Nearest Neighbors, Neural Networks (including Deep Learning), Hybrid Methods, and Other Methods. Finally, some conclusions about the state of the art are included.

2.1 k-Nearest Neighbors

k-Nearest Neighbors (kNN) is one of the most used family of methods for traffic forecasting. Its simplicity and low tuning requirement favour kNN as an early used machine learning method in this field and frequently is included in more recent studies for comparative purposes. Although in the last years more sophisticated methods are used to solve this problem, some current works are focused in provide new approaches to improve the results of kNN algorithms.

The recently new kNN approaches are focused on using spatiotemporal information instead of only employing temporal information. To perform this, the spatiotemporal relationships are included in the distance function and/or in the weights of the different neighbors. Cai et al. [2016] proposes an improved kNN model to enhance forecasting accuracy based on spatiotemporal correlation and to achieve multistep forecasting using floating car speed data. The same year, Xia et al. [2016] proposes a Spatial-Temporal Weighted K-Nearest Neighbor model, which considers the spatiotemporal correlation and weight of traffic

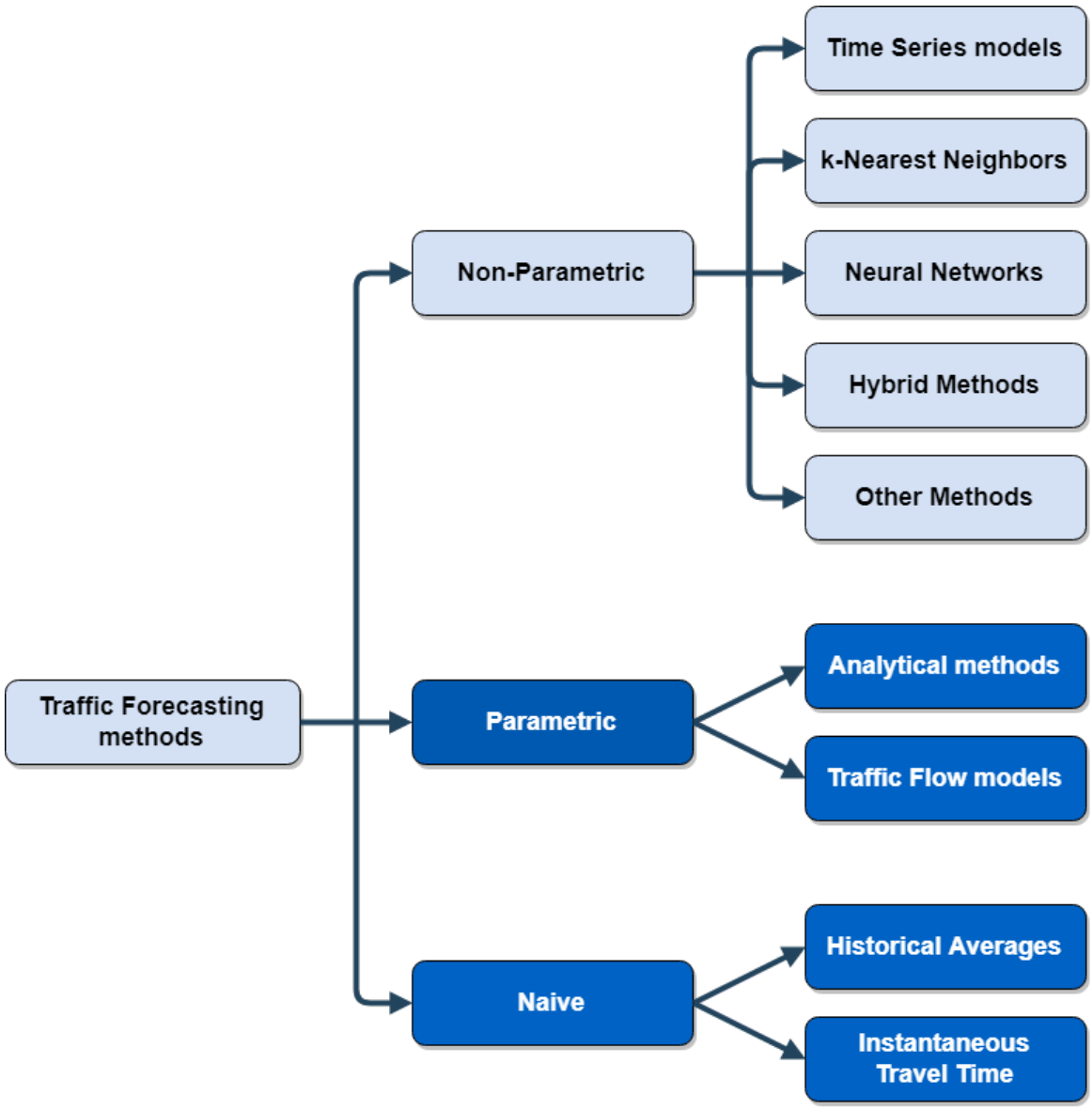


Figure 2.1: Traffic forecasting methods classification. The clear boxes denotes the studied thoroughly methods.

flow with trend adjustment features. This method is implemented on a Hadoop platform for parallel prediction of traffic flow in real time using a big volume of data from taxi trajectories. Finally, Cheng et al. [2018] proposes a new adaptive Spatio-Temporal K-Nearest Neighbor model (adaptive-STKNN) for short-term traffic forecasting. Four prediction models, including Cai et al. [2016], are compared with the adaptive-STKNN model and the results show that the adaptive-STKNN model outperforms the others.

Another interesting work related to the kNN methods is presented by Sun et al. [2017]. Instead of trying to improve the kNN method, it analyzes different kNN strategies in order to organize them and select the right one to improve prediction accuracy. The study is focused in select the parameter strategy, which is the way for the choice of kNN parameter values, and the data strategy, which is the way to separate training dataset to suitable sub-datasets according to different characteristics of instances. Finally, it suggests considering all parameter strategies simultaneously as ensemble strategies especially by including "window size" together with "number of nearest neighbors" and "search step length" in flow-aware strategies.

2.2 Neural Networks

Neural Networks (NN) has become a hot topic recently because of improvements on the algorithms that have been able to solve classical problems in the computer vision area. This motivated the use of these improved algorithms to other areas like traffic forecasting. A NN is an information-processing structure, distributed and parallel, that takes the form of a directed graph where the nodes are neurons and the edges are connections. A neuron is a local computing device that gets an input vector, combines this vector with a vector of local parameters (its weights) and with local information (e.g., the neuron's previous state) and outputs a scalar quantity as a result. The output can be part of the final NN result or can be delivered as part of the input of another connected neuron. Despite some complex NN are considered Deep Learning models, in order to classify the methods in a better way, the Deep Learning methods (including complex NN) are mentioned in a subsection of the NNs section.

In Ye et al. [2012], NNs are purposed to short-term traffic speed forecasting when the collected GPS data is recorded at irregular time intervals and the method is tested with the GPS data from 480 taxis. In the case of Zheng and Van Zuylen [2013], NNs are used to estimate the complete link travel times using the partial link or route travel times obtained from the collected traffic data of some simulated probe vehicles. Also Fusco et al. [2015] suggests this kind of models to perform prediction of road link speed on urban road networks. In this study, the method is tested and compared with a Bayesian Network approach using a floating car dataset, concluding that any of the two methods is superior respect to the other.

Besides basic NN models, some variations are presented in the current literature in order to improve the results. Sun et al. [2012] combines Graphical Lasso (GL) to building a sparse graphical model in order to extract the informative historical traffic flows and NN to perform the prediction. The paper compares this method with a Gaussian Process Regression and the results show the superiority of the GL-NN

model. In Wang et al. [2016], a novel Space-Time Delay Neural Network model (STDNN) is purposed for travel time prediction. The model is able to accommodate the heterogeneous space-time autocorrelation of a road traffic network and in the experiments the STDNN model perform better than the remaining benchmark models. The Raza and Zhong [2017] study uses Genetic Algorithms (GA) to optimize the input datasets for NN models and Locally Weighted Regression (LWR) models. These models are tested to predict 5-minute short-term traffic speed for four lanes of an urban road and the experiments show that GA-LWR is more accurate than GA-NN.

2.2.1 Deep Learning

Due to the increment in the quantity and the diversity of data generated, new more complex methods make sense in order to exploit to the fullest this data. The computational advances in the last years and the development of new software, focused to use and optimize this kind of methods, makes computationally more demanding models possible. This is the case of Deep Learning (DL) algorithms, which are becoming more used in most fields where the machine learning methods are applied. Also in the traffic forecasting field, where the researchers that apply DL to this problem is increasing in the last times.

A clear example of the developed software to use DL models is TensorFlow. Yi et al. [2017] is the first research that applies the TensorFlow DL NN model to the estimation of traffic conditions. The model is able to estimate the traffic flow conditions using real-time GPS traffic data with an accuracy of 99% according to their authors. In Polson and Sokolov [2017] a deep learning model is developed to predict traffic flows and it is tested with two cases that have sharp and very suddenly traffic flow regime changes. The paper shows that deep learning architectures can capture the nonlinear spatiotemporal effects of traffic flows and provides precise short-term traffic flow predictions.

The Long Short-Term Memory (LSTM) methods are a subfamily of the Recurrent Neural Networks (RNN) which is capable of learning long-term dependencies, remembering information for long periods of time. This kind of models are used in studies of time series and other sequential data and indicates their advantage properties. Also in the traffic forecasting field, the LSTM models have been applied. In Yanjie Duan et al. [2016], the LSTM NN model is explored for travel time prediction and the evaluation results show that travel time prediction error is relatively small (around 7.0%). The Liu et al. [2017] study establishes a set of LSTM NN with Deep Neural Layers (LSTM-DNN) using 16 settings of hyperparameters. Then competitive LSTM-DNN models are extracted and tested using data collected from loop detectors. The results from the performed comparison along with some linear and DNN models demonstrate the advantage of LSTM-DNN models. Also in Du et al. [2018], an LSTM based regression model is purposed to predict 24-hour traffic counts data. From the simulation of single location results, it can find out the model can well predict patterns of volume curves, such as peak, valley, ascent and decent. In order to offer a different point, Fu et al. [2017] compares an LSTM model with a Gated Recurrent Units (GRU) model, which is a NN method similar to LSTM suggested by Cho et al. [2014]. The methods are used to predict short-term traffic flow with the data collected from over 15,000 sensors. The results of the experiments conclude that, as expected, GRU and LSTM outperforms the ARIMA model. Also, the results show that

GRU outperforms LSTM in this problem.

Another subclass of Deep Neural Networks methods is the Convolutional Neural Networks (CNN). The CNN is an efficient and effective image processing algorithm with remarkable results achieved. In this way, Ma et al. [2017] introduces an image-based method that represents network traffic as images, and employs a CNN to extract spatiotemporal traffic features contained by the images. The effectiveness of the proposed method is evaluated by taking two real-world transportation networks and the results show that the proposed method outperforms other algorithms such as kNN, NN, RNN, and LSTM. Although a standard CNN for regular grids is clearly not applicable to general graphs, the researchers explore how to generalize CNN to structured data forms. Yu and Yin et al. [2017] presents a novel DL framework, called Spatio-Temporal Graph Convolutional Networks (STGCN), which is a CNN that treats the problem in a graph formulation. Experiments show that the model outperforms other state-of-the-art methods (such as LSTM and GRU) on two real-world datasets.

Besides these different types of DL methods, in the literature can be found diverse studies where different approaches are merged. For example, Cheng et al. [2017] propose an end-to-end framework called DeepTransport, in which CNN and RNN are utilized to obtain spatial-temporal traffic. The presented experiments demonstrate that the method captures the complex relationship in the spatiotemporal domain and outperforms traditional and a state-of-the-art DL method. Also, Yu et al. [2017] present a spatiotemporal recurrent convolutional networks model (SRCN), which take as input a set of static images that represents the network-wide traffic speeds. The method inherits the advantages of CNN (captures the spatial dependencies of network-wide traffic) and LSTM (learns temporal dynamics). The performed experiments show that the method outperforms other deep learning-based algorithms in both short-term and long-term traffic prediction (which is interesting because it is a less studied modality). Another approach that merges CNN and LSTM is proposed in Cui et al. [2018], named High-Order Graph Convolutional Long Short-Term Memory Neural Network (HGC-LSTM). This use CNN applied to graphs instead of to images. Also, it presents a novel Real-Time Branching Learning algorithm for the HGC-LSTM framework to accelerate the training process for spatiotemporal data. Experiments show that the HGC-LSTM network consistently outperforms state-of-the-art baseline methods (such as NN, LSTM and LSTM applied to graphs) on two heterogeneous real-world traffic datasets.

2.3 Hybrid Methods

In the literature is extended the fact that the best method for all the situations doesn't exist, so a model selection is needed to find the most appropriate algorithm and configuration. A practical alternative is to provide a model, algorithm or heuristic approach to combine predictions. Combining prediction techniques is a tendency that explores the combination of different models to improve their individual accuracy, and more recently combining the prediction method with techniques to preprocess data and to optimize the model. Moreover, the risk of combining forecasts is lower than the risk of choosing a single model with increased spatiotemporal complexity.

In Xu et al. [2015] is proposed an online framework that could learn from the current traffic situation (or context) in real-time and predict the future traffic by matching the current situation to the most effective prediction model trained using historical data. The base predictors used are 6 Naive Bayes models for 6 representative situations and the presented experiments show that the proposed approach significantly outperforms other studied solutions. Instead of combining the same kind of models, Mendes-Moreira et al. [2015] use three different algorithms (Random Forest, Projection Pursuit Regression and Support Vector Machines). The results conclude that the ensemble approach is able to increase accuracy but this solution increases the complexity of the system. As it has been mentioned before, some hybrid approach is focused on using additional algorithms to optimize the prediction model, this is the case of Pan and Shi [2017]. It presents a hybrid model based on Grey Neural Network (GNN) and Fruit Fly Optimization Algorithm (FOA), which can exploit the characteristics of grey system model requiring less data, the non-linear map of NN and the quick-speed convergence of FOA, and has a simpler structure. It improves the performance on short-term traffic forecasting of single GNN model and the GNN model with Particle Swarm Optimization. In Guo et al. [2018] a fusion-based framework is developed to improve the accuracy of traffic prediction based on a combination of multiple standalone predictors that work accurately under different traffic conditions. The study is focused to evaluate three different fusion strategies (averaged, weighed and kNN methods) applied to three different machine learning methods (NN, Support Vector Regression and Random Forests). The conclusions show that suitably configured fusion based methods improve final prediction results and that the kNN fusion based method can achieve significantly superior results.

2.4 Other Methods

In the literature there are other existing machine learning techniques that has been used for the traffic forecasting. One of the simplest but very used approaches is the Bayesian Networks (BN). This type of networks represents a set of variables in function of the probabilistic relationships with the others. So, these network-based models are used in traffic problems because they can be formulated to reflect the topology of the network and then to exploit larger correlations between measures collected on close links. Khan [2012] reports a Bayesian predictive travel time information system that fuses real-time data with other factors that influence travel time. An example application is included and the results obtained show that the system is able to produce the pattern of travel time condition that is likely to be experienced by travelers. Fusco et al. [2015] compares a BN and an NN model for prediction of road link speed using two large floating car dataset. As is mentioned in the NN section, the two models exhibit similar performances. In Zhu et al. [2016], a Linear Conditional Gaussian BN model is used to consider both spatial and temporal dimensions of traffic as well as speed information for short-term traffic flow prediction. The results show that the prediction accuracy will increase significantly when both spatial data and speed data are included. The Scalabrin et al. [2017] study tackle the problem through a dedicated BN for each road, which is utilized to capture the spatiotemporal relationships underpinning traffic data from nearby road links. The forecasting capability and anomaly detection accuracy of the proposed framework is tested using a large dataset from a real network deployment and the results show the great capabilities

of the proposed framework.

Another machine learning algorithm used is the Support Vector Machine (SVM). Xu et al. [2016] proposes a spatiotemporal Variable Selection-based Support Vector Regression (VS-SVR) model fed with the high-dimensional traffic data collected from the available road segments. The results indicate that the VS-SVR model is an effective approach for short-term traffic volume prediction in a complex urban road network. A different approach of the SVM, named Least Squares SVM, is applied in Li et al. [2017] for short-term traffic demand forecasting model. The model is tested with the data collected from a taxi booking mobile app (Didi Dache app) and the conclusions highlight its prediction performance and the good capability of capturing the non-stationary characteristics of the short-term traffic dynamic systems.

The Decision Trees (DT) are a simple machine learning method based on stratifying or segmenting the predictor space into a number of simple regions. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as DT methods. Although a single DT sometimes is too simple to solve complex problems, some methods use a combination of a set of them to increase the capability of a single one. One of these methods is the Gradient Boosting Machine (GBM), which build DT one at a time, where each new tree helps to correct errors made by the previously trained tree. Zhang and Haghani [2015] employs a GBM model to analyze and model freeway travel time to improve the prediction accuracy and model interpretability. The proposed method is compared with another popular ensemble method and the study results indicate that the GBM model has its considerable advantages in freeway travel time prediction. Another method that combines DT is the Random Forest, train each tree independently using a random sample of the data. This method is used in Moreira-Matias et al. [2016] as an offline model for travel time prediction. Another approach of the DT is the Incremental Decision Trees, which can be built adding instance by instance without store the full dataset. They are very useful when the whole dataset is not available at the start time or the dataset is too big to be stored. The Wibisono et al. [2016] use a Fast Incremental Model Trees-Drift Detection to predict and visualize the traffic conditions in a particular road region where the size of data is very huge. An incremental experiment is performed in order to show the evolution of the error when more and more instances are added.

As has been mentioned at the beginning of this section, many studies in the literature regarding short-term traffic flow prediction use Autoregressive models, which are linear estimators based on the past values of the modeled time series. Although these methods have lost popularity in recent times, some new works propose different approaches to improve its effectiveness. Abadi et al. [2015] includes a prediction algorithm based on an autoregressive model that adapts itself to unpredictable events. As a case study, the algorithm predicts the flows of a traffic network in San Francisco, CA, USA, using a macroscopic traffic flow simulator. The simulations demonstrate the accuracy of the proposed approach for traffic flow prediction, with errors from 2% for 5-min windows to 12% for 30-min windows even in the presence of unpredictable events. Another approach of the autoregressive models is the ARIMA models. Concretely, Salamanis et al. [2016] use a Space-Time ARIMA, including state-space approach. It introduces a graph-theory-based technique for managing spatial dependence between roads of the same network and selects the most correlated roads to perform the model. Benchmark results indicate improvements on the

computational time required and that it works better in different network topologies. In Schimbinschi et al. [2017], a learning Topology-Regularized Universal Vector Autoregression (TRU-VAR) is proposed for traffic forecasting in large urban areas. The system produces reliable forecasts in large urban areas and it is described as scalable, versatile and accurate.

Aside from the previously mentioned methods, some additional methods have been used in the literature. Some examples are: Sun and Zhang [2007] presents the Selective Random Subspace Predictor to deal with incomplete data; like the previous one, Haworth and Cheng [2012] uses a Kernel Regression to perform traffic forecasting under missing data assumption; Wu et al. [2016] proposes a Spatio-Temporal Random Effects to reduce the computational complexity of the problem; in Bezuglov and Comert [2016] different approaches of three Grey System theory models are tested for short-term traffic speed and travel time predictions; and the Chen et al. [2017] study the application of a Fuzzy Markov Chain model to short-term traffic prediction from GPS data.

2.5 Summary and Conclusions

This Chapter presents the state of the art in traffic forecasting. Concretely, the included works are mostly located between 2004 and 2018 and contains papers that perform traffic forecasting in urban contexts and/or uses FCD as the data source to perform the predictions. The publications revised are classified by the most common methods found in the literature: kNN, NN (including DL), Hybrid Methods, and Other Methods. Following, some conclusions about the previous Sections are presented.

Due to the simplicity of the kNN, they were an early tested machine learning method in traffic forecasting and frequently used for comparative purposes. The new kNN approaches are focused on using spatiotemporal information. Among the presented options, the adaptive-STKNN proposed by Cheng et al. [2018] highlights because it fits very well with the focus of this work (urban network and FCD) and it presents good results. Additionally, could be interesting apply the analized parameter and data strategies by Sun et al. [2017] to try to improve the results of Cheng et al. [2018].

As in the kNN case, the simple NN are used in order to compare and quantify the improvement of new algorithms. This does not mean that new NN approaches do not appear to perform traffic forecasting. Wang et al. [2016] proposes the STDNN, which is able to accommodate the heterogeneous space-time autocorrelation of a road traffic network. Also, it is interesting the Raza and Zhong [2017] study that uses GA to optimize the input datasets and reduce the computational cost of the model. Although these models outperforms many of the typical methods, the evolution of the NN models to the DL ones is not an exception in the traffic forecasting field.

Currently, most of the newly proposed methods for traffic forecasting uses a DL approach. The DL architectures can capture the nonlinear spatiotemporal effects of traffic flows and provides precise short-term traffic flow predictions, as it is shown in Polson and Sokolov [2017]. Inside of the DL methods, one of the most studied methods is the LSTM-DNN (including GRU, which is considered a variation

of the LSTM), which works very well with time series data because the model structure can remember information from past events. Fu et al. [2017] compares the LSTM model with a GRU model and it concludes that GRU outperforms LSTM in this problem. Another DL approach is the use of CNN. The CNN is an efficient and effective image processing algorithm with remarkable results achieved. Similarly, Yu et al. [2017] presents a novel DL framework, called STGCN, which is a CNN model that treats the problem in a graph formulation and outperforms the tested LSTM and GRU methods. Joining these two approaches, the SRCN method proposed by Yu et al. [2017] and the HGC-LSTM method proposed by Cui et al. [2018] are able to inherits the advantages of CNN (captures the spatial dependencies of network-wide traffic) and LSTM (learns temporal dynamics). Although they use different input formats (images and graphs respectivelly), both outperforms CNN and LSTM alternatives in short-term and long-term traffic prediction. In general, DL methods are the most frequent in the novelty literature and show the best results in traffic forecasting. Based in the forecasting capabilities, the models wich join CNN and LSTM approaches are the most insteresting to test. But, the LSTM model can be also interesting to use given that it does not require the network topology.

In the literature, it is extended the fact that the best method for all the traffic forecasting situations does not exist, making necessary a model selection to solve the problem. Another practical alternative is to provide a hybrid model, algorithm or heuristic approach to combine predictions. Xu et al. [2015] and Mendes-Moreira et al. [2015] propose a combination of models in order to improve the results in the traffic forecasting. Another hybrid approach, such as Pan and Shi [2017], is focused on using additional algorithms to optimize the prediction model. In Guo et al. [2018] three different fusion strategies applied to three different machine learning methods are tested in order to perform better hybrid models.

Out of the general families classified, some other methods highlight and have to be considered. From the BN approach, Scalabrin et al. [2017] study tackles the traffic forecasting problem through a dedicated BN for each road, which is utilized to capture the spatiotemporal relationships between nearby links. The result offers a great interpretability of the final models and a good accuracy in the predictions. Inside of SVM methods, the spatiotemporal VS-SVR model proposed in Xu et al. [2016] fed with the high-dimensional traffic data collected from the available road segments, and the results shown its effectivity for short-term traffic volume prediction in a complex urban road network. Another interesting model, from the Autoregressive models family, is the TRU-VAR proposed in Schimbinschi et al. [2017]. The model produces reliable forecasts in large urban areas and it is described as scalable, versatile and accurate.

From the revised literature it is clear that many interesting methods exists in order to perform traffic forecasting. Although typically this problem has been treated as a time series problem, an important part of the most recent researches are focused in combine spatial and temporal information in order to improve the prediction performance. Especially in urban contexts, where the traffic is not as temporally recurrent as in non urban scenarios. In this way, the most atractive methods from the analyzed ones are: adaptive-STKNN proposed by Cheng et al. [2018], the SRCN method proposed by Yu et al. [2017] and the HGC-LSTM method proposed by Cui et al. [2018]. Also, the possibility of develop an hybrid method in order to improve the single ones by separate is very interesting.

Taking account the current trending of the traffic forecasting researches, the limited scope of the

project and the author interests, this project will be focused in DL approaches. Due to it is does not exists a generic suite to test this kind of solutions in the same conditions, it is so difficult to compare different methods from the results of the original papers. So, the proposal is to develop the SRCN and the HGC-LSTM methods to perform traffic forecasting in urban scenarios using floating car data and compare it. The LSTM model will also be included in order to use it as baseline model and evaluate the improvement of the others. Finally, based in the work of Fu et al. [2017], which concludes that the GRU method outperforms the LSTM method in traffic forecasting, the GRU method will also be developed in this project as alternative of LSTM.

Chapter 3

Neural Networks required background

As it is explained in Section 2.5, this project is focused in Neural Networks methods, concretely in Deep Learning ones. This Chapter describes what are the Neural Networks and two of their most important families: Recurrent Neural Networks and Convolutional Neural Networks. Finally, their most relevant parameters are also introduced.

3.1 Neural Networks preliminary concepts

A Neural Network (NN) is a type of machine learning methods slightly inspired in the behavior observed in the axons of neurons in the brains. Formally, a NN is an information-processing structure that takes the form of a directed graph where the nodes are neurons and the edges are connections. A neuron is a local computing device that gets an input vector, combines this vector with the local parameters (weights and bias) and outputs a scalar quantity as a result. The output can be part of the final NN result or can be delivered as part of the input of another connected neuron.

A simple neuron can be defined in a more detailed way separating its behavior in three parts:

- **Input:** The input of a neuron is composed of three elements:
  - $x \in \mathbb{R}^n$: the vector of inputs from the previous n neurons.
  - $w \in \mathbb{R}^n$: the vector of weights of the neuron inputs.
  - $b \in \mathbb{R}$: the bias of the neuron.
- **Input combination:** All the inputs are combined to compute the neuron state, which is the input

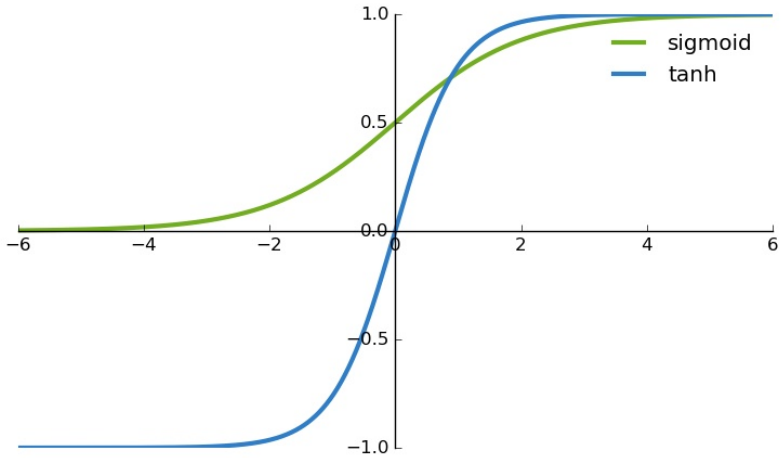


Figure 3.1: A graphical representation of the sigmoid function and the hyperbolic tangent (tanh) function. *Source:* `http://ronny.rest/blog/`

of the activation function. Normally, the neuron state z is computing with the Equation 3.1.

$$z = \sum_{i \in n} w_i x_i + b \tag{3.1}$$

- **Activation function:** Finally, an activation function $f()$ is applied to the result of the previous combination and the result of this function is the output y of the neuron. Typically, the sigmoid and the hyperbolic tangent (tanh) functions are the most used (Figure 3.1).

Figure 3.2 shows a generic schema of a neuron with the three mentioned parts.

The basic structure of a NN is composed of different layers of neurons. Typically, the neurons of a layer are only connected to the neurons of the next layer. It exists three different kinds of layers:

- **Input layer:** The first layer of a NN is the input layer, which represents the input of the model. Usually, this layer is not counted given that it does not perform any operation and its output is simply the input.
- **Output layer:** The last layer of a NN is the output layer, which computes the final result of the model using a neuron for each output variable.
- **Hidden layer:** The remainder layers are the hidden layers.

Figure 3.3 shows the basic schema of a simple NN with the three different mentioned types of layers. A NN with more than one hidden layer is considered a Deep Neural Network.

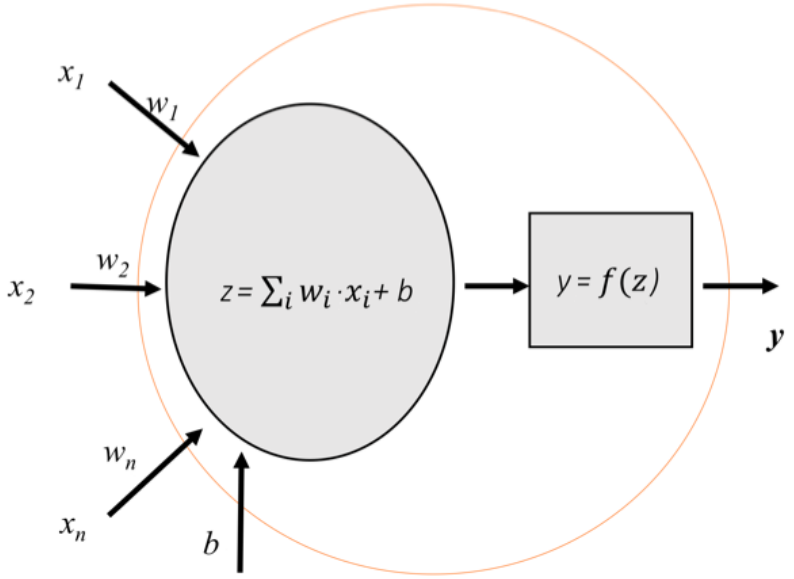


Figure 3.2: Basic schema of a simple Neuron. *Source:* `https://torres.ai/`

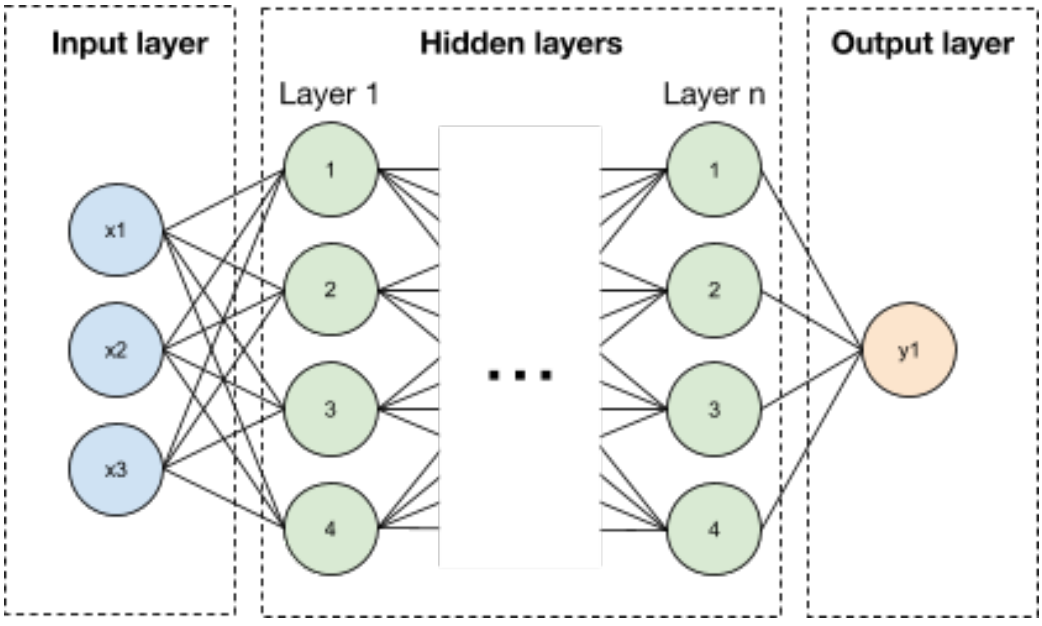


Figure 3.3: Basic schema of a simple Neural Network. *Source:* `http://mindwise-groningen.nl/deep-learning-the-beautiful-mind/`

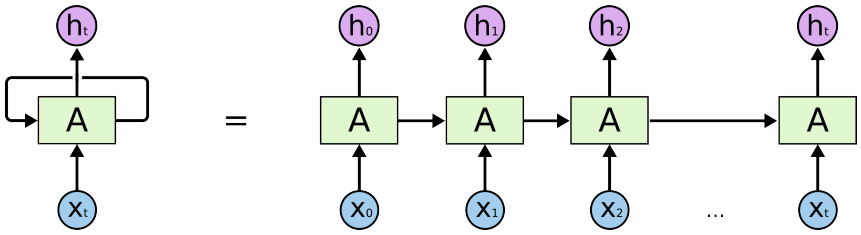


Figure 3.4: Schema of an unrolled Recurrent Neural Network. *Source:* `http://colah.github.io/`

3.2 Recurrent Neural Networks

The networks whose directed graph does not contain cycles are considered feed-forward Neural Networks (this is the traditional neural networks approach). Against, the networks whose graph contains cycles are called Recurrent Neural Networks (RNN). Although it seems a completely different schema, if the cycle is unrolled the RNN can be considered as multiple copies of the same network, each passing a message to a successor. Figure 3.4 shows an unrolled RNN diagram where A represents the repeating part of NN and x_t is some input that outputs a value h_t.

Contrary to the human thinking process, the feed-forward Neural Networks start each process from a scratch state. Many problems related to sequences and lists can use the previous states in order to improve the results for the next ones. In this way, RNN allows information to be passed from one step of the network to the next one, achieving to persist information between instances. Some application examples of these models are based on problems of temporal series, where this recurrent feedback takes advantage of the temporal relation between the data.

3.2.1 Long Short-Term Memory Neural Networks

Theoretically, the simple RNNs are capable of handling "long-term dependencies", but in practice, they do not seem to be able to learn them. To deal with this problem Hochreiter and Schmidhuber [1997] propose Long Short-Term Memory Neural Networks (LSTM), which are a kind of RNN capable of remembering information for long periods of time. The difference between the simple RNN and the LSTM is the structure of the repeating module. In standard RNNs, this repeating module will have a very simple structure, such as a single hyperbolic tangent layer. However, as it is shown in Figure 3.5, in LSTMs instead of having a single neural network layer, there are four. In order to ease the understanding of the schemas, the notation defined in Figure 3.6 is used.

The main peculiarity of the LSTMs is the double recurrent connections in the repeating module. Besides the typical recurrent connection in RNNs which reuse the output of the previous prediction, the LSTMs has the cell state (represented in the figure 3.5 by the top horizontal arrow). This cell state allows keeping cumulative information which can be modified in each execution through some minor linear interactions. The internal LSTM behavior can be explained in 4 steps:

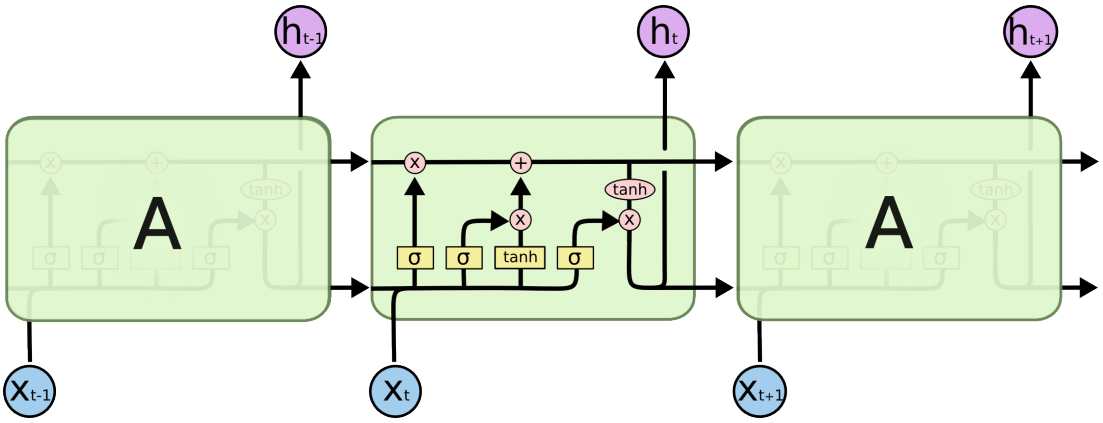


Figure 3.5: Schema of a Long Short Term Memory network layer. *Source:* `http://colah.github.io/`

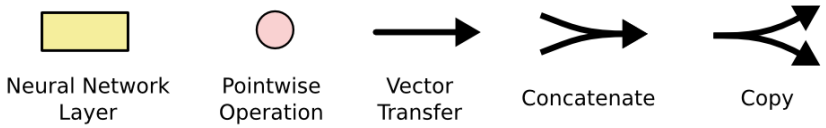


Figure 3.6: Notation of the LSTM schemas. *Source:* `http://colah.github.io/`

1. The first part in a LSTM, named forget gate (Figure 3.7), decides what information remove from the cell state. It uses a sigmoid layer that combines h_{t-1} (the hidden state) and x_t (the input) and outputs a number between 0 (remove it completelly) and 1 (keep it completely) for each number in the cell state C_{t-1} (Equation 3.2).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.2}$$

2. The second one, named input gate (Figure 3.8), selects what information has to be added to the cell state. It is formed by two parts. First, a sigmoid layer uses h_{t-1} and x_t and then, it generates a

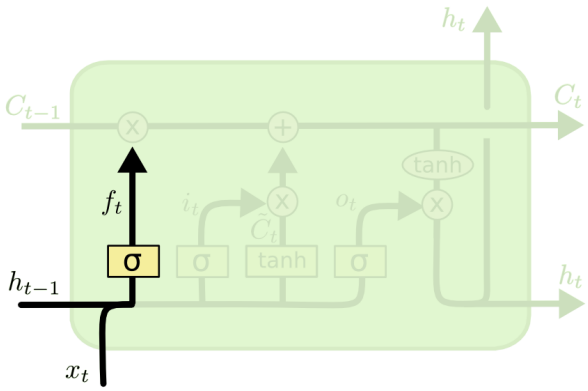


Figure 3.7: The focus of the LSTM schema for the first step. *Source:* `http://colah.github.io/`

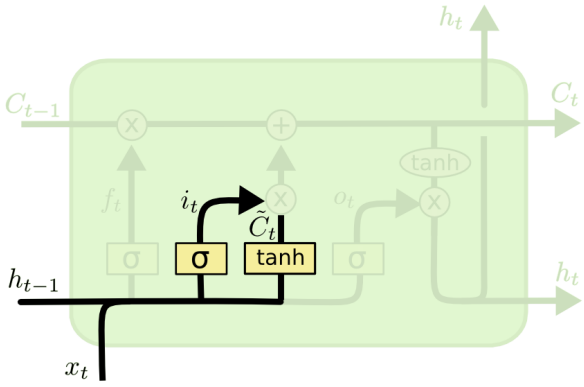


Figure 3.8: The focus of the LSTM schema for the second step. *Source:* `http://colah.github.io/`

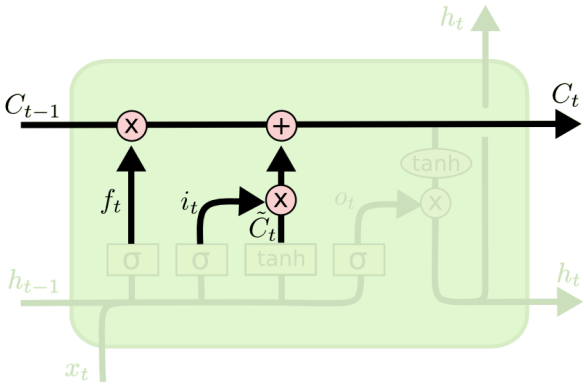


Figure 3.9: The focus of the LSTM schema for the third step. *Source:* `http://colah.github.io/`

vector i_t of values between 0 and 1 (Equation 3.3). These values quantify what portion of the state values will be updated. Next, a tanh layer combines h_{t-1} and x_t to create a vector of new candidate values, $\widetilde{C}_t$ (Equation 3.4).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.3}$$

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.4}$$

3. With the results of the previous steps, the cell state can be updated (Figure 3.9). Multiplying the old state by f_t, the new cell state forgets the information selected in the first step. So, the new information from step two is added to the new state. This new information is obtained by multiply the i_t (which values have to be updated) and $\widetilde{C}_t$ (the new candidates values). The full operation is in the Equation 3.5.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{3.5}$$

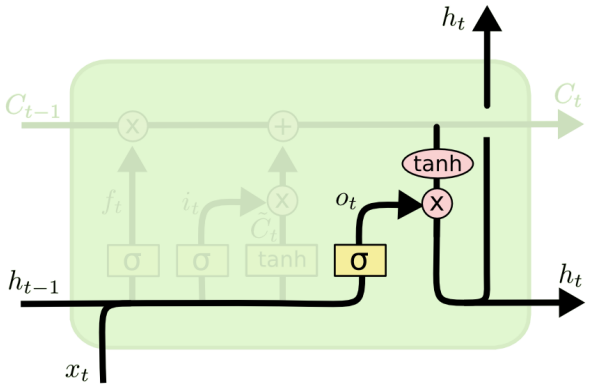


Figure 3.10: The focus of the LSTM schema for the fourth step. *Source:* `http://colah.github.io/`

4. The last step (Figure 3.10) is to compute the output, which is a filtered version of the updated cell state. First, a sigmoid layer decides, based in h_{t-1} and x_t, what parts of the cell state are relevant for the output (Equation 3.6). Then, the cell state is mapped to values between -1 and 1 using a tanh multiplied by the results from the sigmoid layer (Equation 3.7).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.6}$$

$$h_t = o_t * tanh(C_{t-1}) \tag{3.7}$$

Although the described LSTM schema is one of the most adopted ones, it exists many slightly different LSTM schemas. Some of them have their own name and due to its influence in the literature are considered as different methods. One of the most popular versions is the Gated Recurrent Unit (GRU), which is a simplification of the LSTM presented by Fu et al. [2017] (Figure 3.11 and Equation 3.8). The main differences between both are:

- The GRU structure combines the forget and input gates into a single update gate.
- Also, it merges the cell state and hidden state in a single connection.
- The output of the GRU is directly the hidden state instead of being a filtered version.

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\ \widetilde{h_t} &= tanh(W \cdot [r_t * h_{t-1}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t} \end{aligned} \tag{3.8}$$

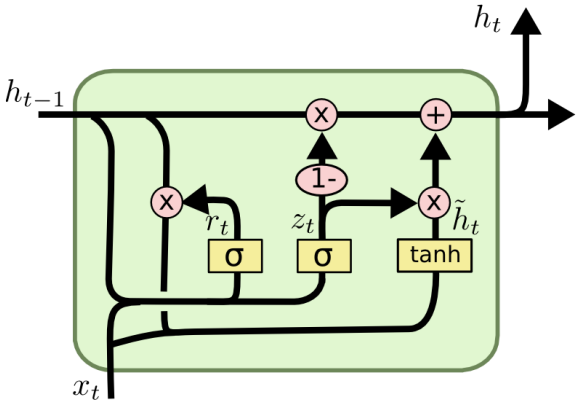


Figure 3.11: GRU schema. *Source:* `http://colah.github.io/`

3.3 Convolutional Neural Networks

Another family of NN is the Convolutional Neural Networks (CNN), which are widely used in machine learning problems when the inputs are images. These networks are composed of an input layer, an output layer, and several hidden layers, some of which are convolutional. More specifically, the two layers that define convolutional networks can be expressed as groups of specialized neurons in two operations: convolution and pooling. Both operations are performed over images, which are inputs with three dimensions (height, width and color channels). In order to simplify the following explanations, the images are supposed greyscale images (one color channel) and consequently, two dimensional inputs.

3.3.1 Convolution operation

The convolutional layers are able to perform convolutional operations to different subregions of the input image. Each neuron of this layer is responsible for applying the operation over a subregion of the image. Intuitively, we can think of a $N \times M$ size window that slides along the entire image. For each position of the window, there is a neuron in the layer that performs a convolutional operation over it. The convolutional operation can be expressed as the scalar product of the subregion of the image ($S \in \mathbb{R}^{N \times M}$) and the weight matrix of the neuron ($W \in \mathbb{R}^{N \times M}$) plus a bias ($b \in \mathbb{R}$). Figure 3.12 shows a simple schema of how a convolutional layer transforms the input.

One of the most important properties of the convolutional networks is that, although each neuron of a layer works over a different subregion of the image, they share the same weight matrix (also named filter or kernel). This reduces drastically the number of intern parameters of the model. The intuitive idea over which this is based is that a filter is able to detect a particular feature in any image subregion. So, normally, a convolutional layer performs multiple filters to each subregion in order to detect multiple features. Following the same example than in Figure 3.12, in Figure 3.13 is shown the output of a convolutional layer with 32 filters of 5×5 pixels.

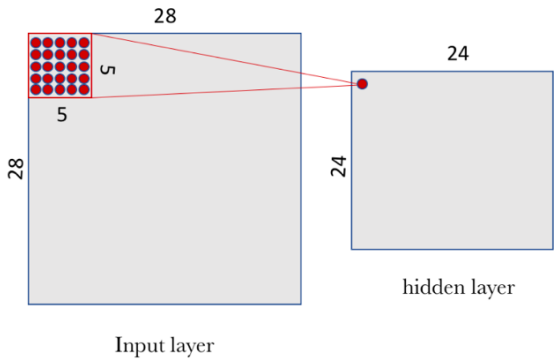


Figure 3.12: Schema of a convolutional operation. *Source:* `https://towardsdatascience.com/@torres.ai`

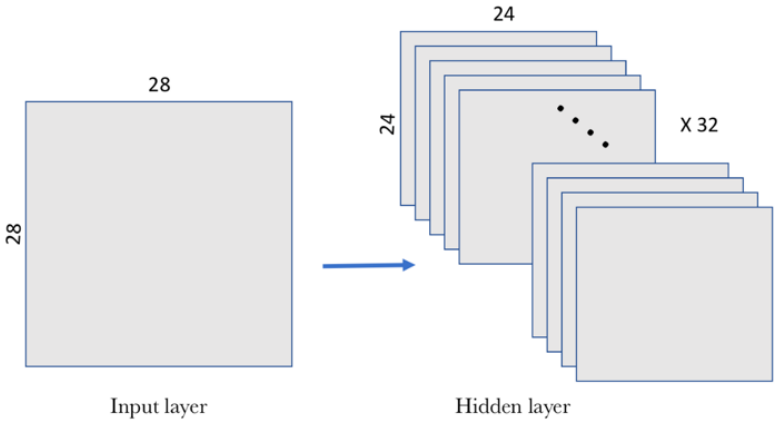


Figure 3.13: Schema of the output of a convolutional layer. *Source:* `https://towardsdatascience.com/@torres.ai`

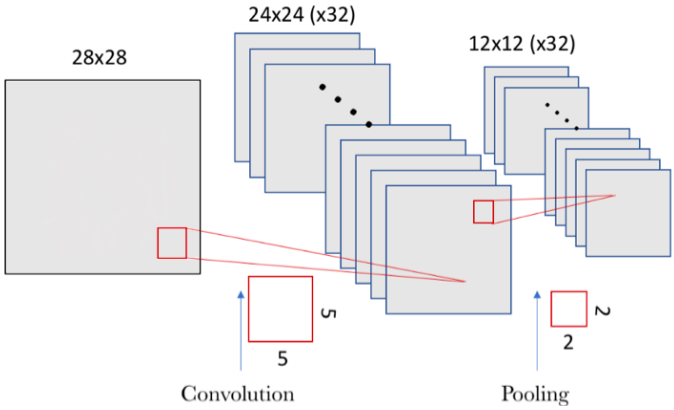


Figure 3.14: Schema of the convolutional and pooling operations application.
Source: `https://towardsdatascience.com/@torres.ai`

3.3.2 Pooling operation

Besides the convolutional layers, the CNN includes also pooling layers, which are usually applied immediately after the convolutional ones. The main function of the pooling layer is to reduce the size of the output of the convolutional layers, but without losing relevant information. The pooling process consists in divide each output matrix of the previous layer in submatrices. So, a pooling function is applied to each submatrix in order to combine all its elements. Typically, the pooling functions are usually the maximum pooling (selects the maximum element of the submatrix) and the average pooling (computes the average of all submatrix elements). In order to complete the example of Figure 3.12, in Figure 3.14 is shown the same example with the application of pooling operation over the 2×2 submatrices.

3.3.3 Convolutional Neural Network structure

Once the individual parts are explained, they can be used to build the final structure of a Convolutional Neural Network. The explanation is based on the example shown in Figure 3.15, where a Convolutional Neural Network model identifies the number represented in an image. The model contains two convolutional layers and two pooling layers that perform the operations explained previously. After this, the model includes a NN that takes as input the extracted features from the last pooling layer. To do this possible, a flatten operation is applied in order to transform the vector of matrices in a vector of numbers. The final NN used in the example is a Full Connected NN, but it can be any kind of NN.

3.4 Neural Network parameters

Like most machine learning methods, the NN has a multitude of configurable parameters that allows to adapt the method to the problem that it wants to solve. These parameters are different for each family

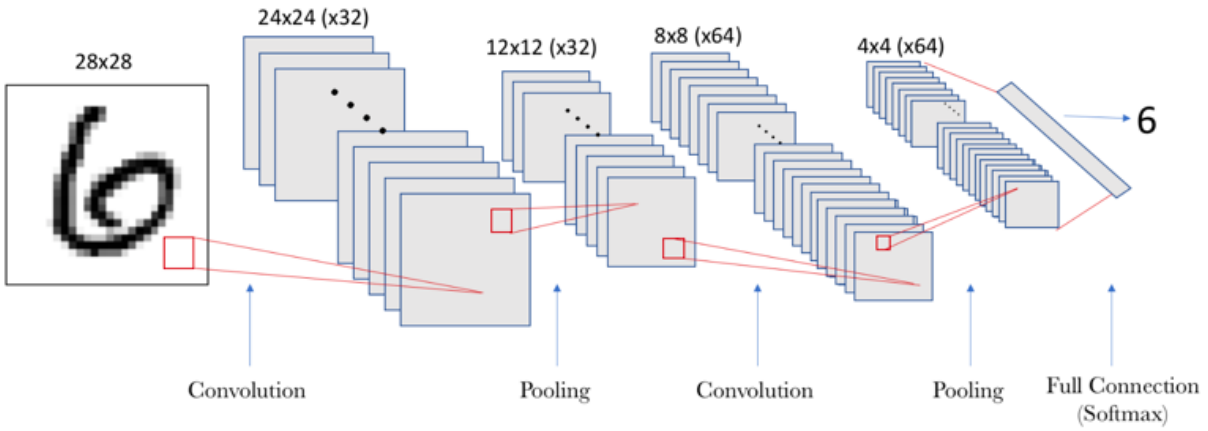


Figure 3.15: Schema of a Convolutional Neural Network model that identifies the number represented in a image. *Source:* `https://towardsdatascience.com/@torres.ai`

of methods and even for methods of the same family. For example, in the NN case, some parameters are common to all the NN models, but LSTM or CNN have their own parameters (in addition to the common ones). The following list defines the most important parameters of NN for this project:

- **Neurons:** The number of neurons of the input layer and the output layer are predefined by the dimension of the data. But the number of neurons that compose the hidden layers can be configured. In fact, each layer may have a different number of neurons.
- **Layers:** In a similar way of the neurons case, leaving aside the input and output layers, the number of hidden layers is also configurable.
- **Activation function:** As it is mentioned in Section 3.1, where the working of a simple neuron is defined, the final output of each neuron is computed by a function called activation function. This parameter specifies what function is used as the activation function.
- **Optimizer:** Like to all the supervised machine learning algorithms, before to use a NN model to predict something, it has to be trained with previous data in order to find the appropriate values for the internal parameters (weights and biases). This training process performs an optimization in order to select the intern parameter values that minimize a loss function (error). So, one of the parameters of the model is which optimization algorithm (optimizer) it will use in the training process.
- **Initializer method:** The initializer method is directly related to the previous parameter. The objective of the optimizer is to find good values for the weights and the biases. This process is done iteratively, updating in each step the selected values. The initializer method is the strategy that is used in the optimization process to select the first initial values for the intern parameters.

As it is mentioned above, the LSTM and the CNN methods have their own parameters. Below are described the most relevant ones for this project.

As it is explained at the end of Section 3.2.1, in the literature exists a lot of different LSTM schemas (for example the GRU method). Some of these different schemas are performed through changes in the structure of the model or in some parameters and a mixture of both (structure and configuration changes). In order to simplify the LSTM use, this work focus on only two parameters of LSTM models (commons to GRU models):

- **Units:** Based on the nomenclature of Section 3.2.1, the units parameter is the size of the cell state of a LSTM layer (vectors C_t and $\widetilde{C}_t$). It does not affect the input nor the output dimension but it increases the number of the model intern parameters.
- **Dropout:** The dropout is the percentage of units randomly selected to be excluded from activation and weight updates while training a network. The objective of this parameter is to deal with the overfitting problem of this kind of models.

It is important to highlight that a LSTM model is composed by LSTM layers with a fixed number of neurons predefined by the structure. So, the neurons parameter it is not used in the configuration of the LSTM models.

On the other hand, the main operations of the CNN (convolution and pooling), described in Section 3.3, are widely configurable in terms of parameters. Some of the fundamentals parameters are the following:

- **Filters:** It defines the number of filters that performs a convolutional layer over the input image.
- **Filter window size:** This parameter is defined by two natural numbers and determines the 2-dimensional size of the subregions of the image where the filters are applied.
- **Strides:** The strides are the number of steps in which the filter window moves (horizontally and vertically) in order to select the next image subregion to perform the convolutional operation.
- **Pooling function:** The pooling function is the function that uses a pooling layer in order to combine all the elements of a submatrix in order to select the output one.
- **Pooling window size:** This parameter is very similar to the filter windows size parameter and determines the size of the submatrix where the pooling operation is applied.

As in the LSTM models case, the number of neurons for the convolutional and pooling layers cannot be determined directly. In the convolutional layers, the number of neurons is implicit to the number of filters, filter window size, and strides. In the pooling layers, the quantity of neurons depends on the pooling window size.

The process of search for the best configuration for the prediction models is named model tuning. This process is one of the most difficult ones on performing a machine learning model because of the number

of configurable parameters (dependent on each method) and the number of possible values for each one (dependent on each parameter and the used implementation). In Section 5.3 the model tuning process is performed for the proposed traffic prediction models using the parameters presented in this Section.

Chapter 4

Development

This Chapter describes the development of the models to perform urban traffic forecasting based on the selected proposals from the state of the art (Section 2.5). Also it includes the details about their implementation and the used tools.

4.1 Proposed models

In this Section, the four selected models to perform urban traffic forecasting are described. Each Subsection is formed by an introduction, the input data preprocessing and the model description. The input data preprocessing is the most complex task, especially for the two last models, because transforming the raw data in a new innovative data representation in order to maximize the data expressiveness for the prediction models. The model descriptions are based on the neural network background included in Chapter 3.

4.1.1 Long Short-Term Memory and Gated Recurrent Unit Neural Networks

This Section explains how the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) methods are applied to the problem described in Section 1.2. As it is shown in Chapter 2, the LSTM is widely used in the literature to perform traffic forecasting. But, based in the work of Fu et al. [2017], which concludes that the GRU method outperforms the LSTM method in traffic forecasting, the GRU method will also be developed in this project as an alternative of LSTM.

Although the LSTM and GRU are analyzed as separated methods, they are explained together because, in fact, the GRU method is a variation of the LSTM method and the main properties are the same. The start point of their development is to transform (or preprocess) the data generated in the simulation phase explained in Section 5.1.

vehicle_id	timestamp	speed(km/h)	section	lane
1	0:02:00	50	1	1
2	0:02:00	40	2	1
1	0:04:00	20	1	1
2	0:04:00	50	1	1
1	0:06:00	35	2	1

period	section 1	section 2
1	40	40
2	null	35

Figure 4.1: Simple example of the transformation between the FCD input and the S corresponding dataset.

Input data preprocessing

The raw data is a collection of records with information about the vehicle identifier, the timestamp of the record, the speed of the vehicle at that moment (km/h) and in which section and lane is the vehicle at that moment. So, this raw data has to be transformed to a new problem representative format. As the objective is to predict the speed of the road sections in a future time period, the data has to contain the speed in the section roads for different time periods. Let $tpd \in \mathbb{R}$ the time periods duration, for example, in minutes. The form of the new dataset S is $S \in \mathbb{R}^{N \times M}$, where N is the number of time windows with duration tdp and M is the number of road sections. So, for a time period i and a section j, S_{ij} represents the speed average of all vehicle records in j during i. Figure **??** shows a very simple example of the input raw data and the corresponding S dataset.

S_{ij} is a missing value when the original data has not any record for a section j at a time period i. Given that the LSTM and GRU methods do not accept missing values, they must be imputed. The imputation of the missing values is an open problem in the machine learning projects. It exists a lot of strategies (less and more complex), but often the most appropriate depends on the nature of the problem. In this case, the missing values are imputed using the k-Nearest Neighbors (kNN) imputation. kNN is an algorithm that is useful for matching a point (row of S matrix) with its closest k neighbors in a multi-dimensional space. The assumption behind using kNN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables. So, the missing value is imputed with the result of a weighted combination of the k closest neighbors. The behavior of this imputation technique requires that the variable with missing values has at least k valid values to interpolate them. This could be a problem if the dataset contains a section without any data for any interval (or less than k). In most machine learning problems, a variable without values is useless and it is removed. But in this case, is a section does not has data reflects that the section is practically always empty, and the forecasting model must provide predictions for this section too. So, for these cases, all the values are filled with the maximum allowed speed of the road section.

Once the missing values are imputed, all the elements of S are scaled to the interval $[0, 1]$. The scaling (or normalization) of the data is a frequent habit in machine learning problems in order to avoid overinfluence of some features depending on their different magnitudes (not relevant in this problem because all the variables are speeds in km/h) and to favor the convergence of the algorithm.

When the data is correctly formatted, it must be split in four different datasets (X_{train}, X_{test}, Y_{train} and Y_{test}) in order to be used. The first split is to separate the input and the output of the model. To explain this, it can be useful to declare a notation where $S = [S_1, ..., S_N]$ and each $S_i = [S_{i1}, ..., S_{iM}]$. So, each S_i refers to row i of the dataset and each S_{ij} is the j-th cell of the row i. Since the objective is to predict the future speeds in the sections given the last ones, given an input S_i the correct output can be defined by S_{i+1}. So, the input matrix $X \in \mathbb{R}^{(N-1)\times M}$ and the output matrix $Y \in \mathbb{R}^{(N-1)\times M}$ can be defined by the Equation 4.1. The input dataset loses a row respect to the S matrix because, given the build process of the X and Y matrices, any row of S can be the output for the last row S_M. To finish with the preprocessing and as usual in machine learning problems, the data must be split also into train and test datasets. The objective of this is to evaluate the resulting model with data that is not used to train it. This avoids getting a wrong good evaluation of overfitted models. So, given a percentage of data to be part of the train (p_train), the data is split into: $X_{train} = X_{1,...,\lceil N*p\_train \rceil}$, $X_{test} = X_{\lceil N*p\_train \rceil+1,...,N}$, $Y_{train} = Y_{1,...,\lceil N*p\_train \rceil}$ and $Y_{test} = Y_{\lceil N*p\_train \rceil+1,...,N}$.

$$\forall i \in (N-1) : X_i = S_i \wedge Y_i = S_{i+1} \tag{4.1}$$

Model description

The behavior of the LSTM and GRU layers are already explained in Section 3.2.1. The specific configuration of these models is explained in Section 5.3, where they are tuned to adapt them to the data. But, in general terms, the LSTM and GRU models can be defined as a sequence of one or more specific layers (LSTM or GRU respectively) connected between them. The last layer of both models (LSTM and GRU) is a fully connected NN layer in order to transform the output of the last LSTM (or GRU) layer to the desired format (in this case, one output by predicted road section speed).

4.1.2 Spatiotemporal Recurrent Convolutional Networks

The Spatiotemporal Recurrent Convolutional Networks model (SRCN) is proposed in Yu et al. [2017]. The main idea of this method is to convert the input in a set of images and use a combination of CNN and LSTM neural networks. According to the authors, this method allows to add the spatial dependencies of traffic in the images and capture them with the CNN layers. Also, the temporal dependencies are learned by the LSTM layers. So, the SRCN model is able to combine spatial and temporal information in order to perform more accurate predictions.

Input data preprocessing

An important part of this model is the generation of the input. It requires to transform the input data samples in images adding the spatial information. This process can be divided into two phases: the representation of the spatial information and the representation of the temporal information.

Figure 4.2: Image of the Camp Nou simulation model.

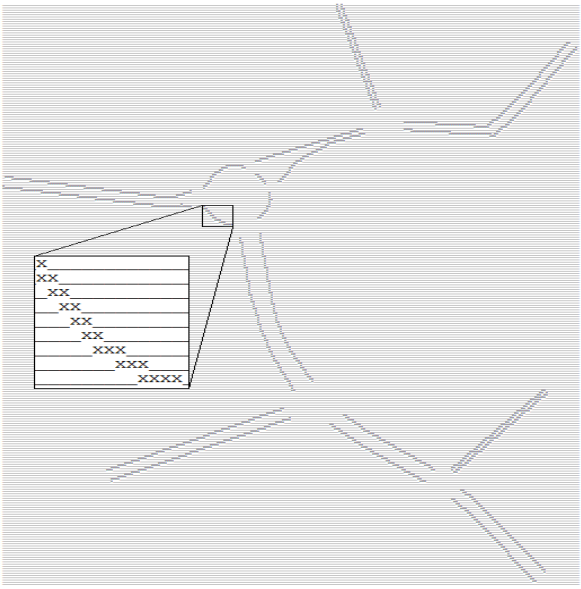


Figure 4.3: Representation of the Camp Nou simulation model printed with 'X' and '_'.

The representation of the spatial information consists in convert the traffic network of the simulation model in a 2-dimensional matrix (like an image). For this, the map of the model is divided into cells (like the pixels of an image) and the road sections of the model are mapped to these cells. Figures 4.2 and 4.3 are an example of an original model and its plain representation. In this example, the representation $R \in \{\text{'}X\text{'}, \text{'\_'}\}^{X \times Y}$ where $R_{ij} = \text{'}X\text{'}$ if exists any road section located in this space and $R_{ij} = \text{'\_'}$ otherwise. The width X and height Y of the matrix are predefined and they control the resolution of the representations ($X = 256$ and $Y = 256$ in the example). As an intuitive example, the previous definition can reflect the main idea, but in fact, to build the input it is necessary to know what road sections are mapped in what cells. So, given the full set of sections called *Sections*, the used representation is a mapping function $sects(R, x, y) = \{s \in Sections : s \textit{ is located in the cell } x, y \textit{ of } R\}$.

After preparing the spatial information of the problem, it must be combined with the temporal one in order to build the input images. This process is based in the matrix $S \in \mathbb{R}^{N \times M}$ generated in the preprocessing of the previous models (Section 4.1.1). As a reminder of the matrix, each cell S_{ij} of the matrix represents the average speed of the section j for a specific time period i. So, for each time period $t \in N$ the SRCN model needs an input image $I^t \in \mathbb{R}^{X \times Y}$ defined in Equation 4.2. The idea is to fill each cell of I with the average of the speeds of the sections that are located in the region represented by the cell. Figure 4.4 contains an example of how the I^t inputs are generated. Also, in the figure 4.5 are shown two examples of the input for the SRCN model. The grey intensity of the lines reflects the state of each road section (average speed).

$$\forall t \in N, \forall i \in X, \forall j \in Y : I^t_{ij} = \frac{1}{|sects(R, i, j)|} \sum_{s \in sects(R,i,j)} S_{ts} \tag{4.2}$$

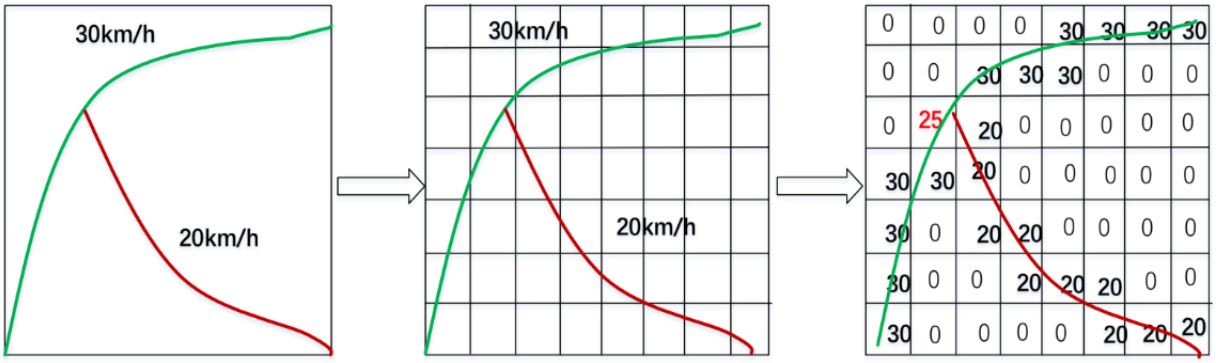


Figure 4.4: Conceptual example of the inputs I^t generation. *Source: Yu et al. [2017]*

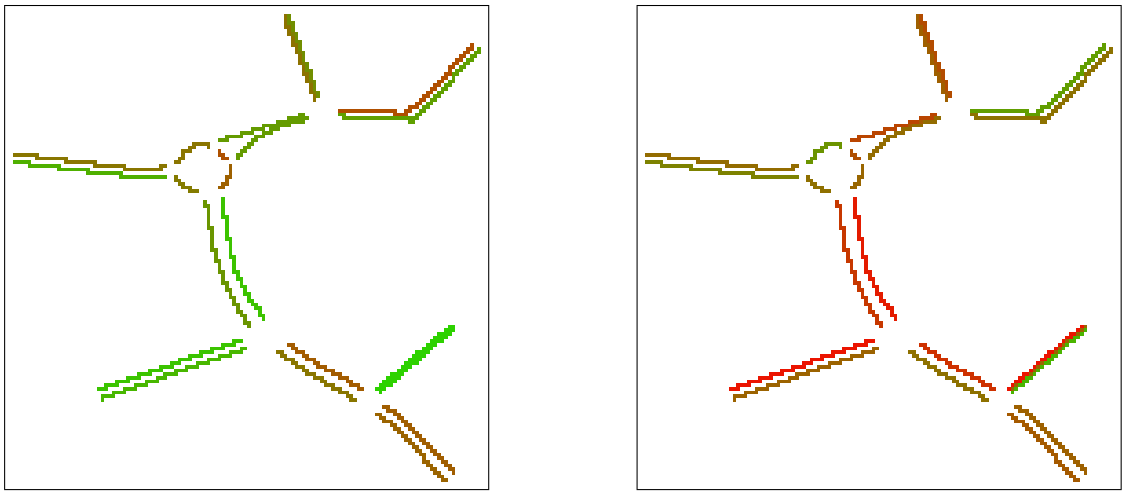


Figure 4.5: Examples of the input images for the SRCN model.

Model description

The SRCN model is a combination of CNN layers (described in section 3.3) and LSTM layers (described in section 3.2.1). Figure 4.6 shows a schema of the SRCN model structure. Obviating the input, the first part of the model is formed by one or more CNN layers. Each CNN layer is composed of a convolutional layer and a pooling layer. Because the format of the output of this first part (2-dimensional samples) is incompatible with the LSTM required format (1-dimensional samples), a flatten layer is added to the final of the last CNN layer. This kind of layers transforms the 2-dimensional samples in 1-dimensional ones chaining all the matrix rows in a single array. After the flatten layer, the LSTM layers can be added in order to incorporate the temporal caption behavior to the model. Finally, like in the simple LSTM model, a fully connected layer is added in order to compute the final output of the full model. The number of CNN layers and the number of LSTM layers are two configurable parameters of the model. These parameters, together with the convolutional filter windows size, the type of pooling, etc. are presented in Section 5.3.

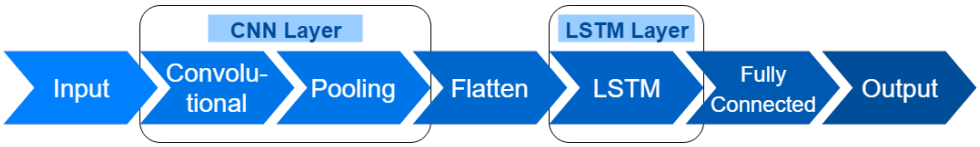


Figure 4.6: Schema of the SRCN model structure.

4.1.3 High-Order Graph Convolutional Long Short-Term Memory Neural Network

The High-Order Graph Convolutional Long Short-Term Memory neural network model (HGC-LSTM), is proposed in Cui et al. [2018]. The background idea is the same that in the SRCN model: perform a model able to capture the spatiotemporal information of the data. The paper describes a way to combines the temporal data and the network graph topology to generate a 3-dimensional dataset. Also, it explains a custom method to transform this dataset into a 2-dimensional input dataset in order to use an LSTM model. The model developed in this project uses the proposed technique to combine temporal data and spatial data but, without performing the method to transform 3-dimensional data into 2-dimensional data. Instead this, and like to in the SRCN model, it uses a combination of CNN and LSTM that accepts 3-dimensional input. The input data preprocessing is explained in depth in the following Subsection.

Input data preprocessing

The processing of the data for this model is based in the matrix $S \in \mathbb{R}^{N \times M}$ explained in Section 4.1.1 and in a graph representation of the network $G = (V, E)$. Where, maybe contrary to the intuitive way, the vertices V represents the M sections and each, given two vertices $v_i, v_j \in V$, the edge $(v_i, v_j) \in E$ represents a turn that allows the vehicles to move from v_i to v_j. So, it exists the adjacency matrix of this graph $A \in \{0,1\}^{M \times M}$ in which each element $A_{ij} = 1$ if the edge (v_i, v_j) exists in the graph and $A_{ij} = 0$ otherwise ($A_{ii} = 0$). And, based in the adjacency matrix, a function $d(v_i, v_j)$ can be defined as the minimum number of edges traversed from v_i to v_j. With this, the k-th order adjacency matrix $\widetilde{A}^k \in \{0,1\}^{M \times M}$ can be defined by Equation 4.3. Using the matrix exponentiation, the matrix $\widetilde{A}^k$ also can be defined by the formula $\widetilde{A}^k = Bi((A+I)^k)$ where $Bi(\cdot)$ is a function to clip the values of all elements of $(A+I)^k$ to 1. Figure 4.7 contains an example of $\widetilde{A}^1$ and $\widetilde{A}^2$.

$$\widetilde{A}^k_{ij} = \begin{cases} 1, & \text{if } d(v_i, v_j) \leq k \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

A first approach of the k-th column of the matrix representation of the spatiotemporal information $TS^k \in \mathbb{R}^{M \times 1}$ could be $TS^k = \widetilde{A}^k \cdot S_t$ for a specific time period $t \in N$. This representation presents a problem which is: although the k-adjacency matrices define the sections that can be reached from each one in k steps, this reachability depends also on the properties of the sections traversed and the time

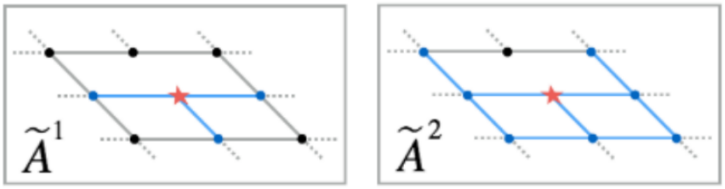


Figure 4.7: Graphical example of $\widetilde{A}^1$ and $\widetilde{A}^2$ matrices. *Source: Cui et al. [2018]*

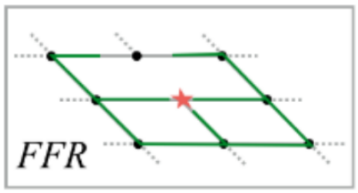


Figure 4.8: Graphical simple example of a FFR matrix. *Source: Cui et al. [2018]*

prediction horizon pt. To deal with this, the free-flow reachable matrix $FFR \in \{0,1\}^{M \times M}$ is defined in the formula 4.4. To define the matrix $T^{FF} \in \mathbb{R}^{M \times M}$, some concepts are introduced in the following points:

- $\forall v \in V : length_v$ represents the length of the section v.
- $\forall v \in V : speed_v^{FF}$ represents the free-flow speed of the section v.
- $\forall (v_i, v_j) \in E : t(v_i, v_j) = \frac{1}{2}\left(\frac{length_i}{speed_i^{FF}} + \frac{length_j}{speed_j^{FF}}\right)$. The $t()$ function represents the free-flow travel time between two adjacent sections i and j.

So, let $t(v_i, v_j)$ the cost of the edge (v_i, v_j) and $sp(v_i, v_j)$ the shortest-path between v_i and v_j, the T^{FF} matrix is defined in Equation 4.5. Figure 4.8 shows an example of a FFR matrix.

$$FFR_{ij} = \begin{cases} 1, & \text{if } T_{ij}^{FF} \leq pt \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

$$T_{ij}^{FF} = \begin{cases} \text{cost of } sp(v_i, v_j), & \text{if exists a path between } v_i \text{ and } v_j \\ \infty, & \text{otherwise} \end{cases} \tag{4.5}$$

Incorporating the new FFR matrix, the k-th column of the matrix representation of the spatiotemporal information can be computed by $TS_t^k = (FFR \odot \widetilde{A}^k) \cdot S_t$ where $\odot$ is the element-wise product of matrices. So, given a predefined maximum K, the representation of the spatiotemporal information for a time period $t \in N$ is $TS_t \in \mathbb{R}^{M \times K} = [TS_t^1, ..., TS_t^k]$.

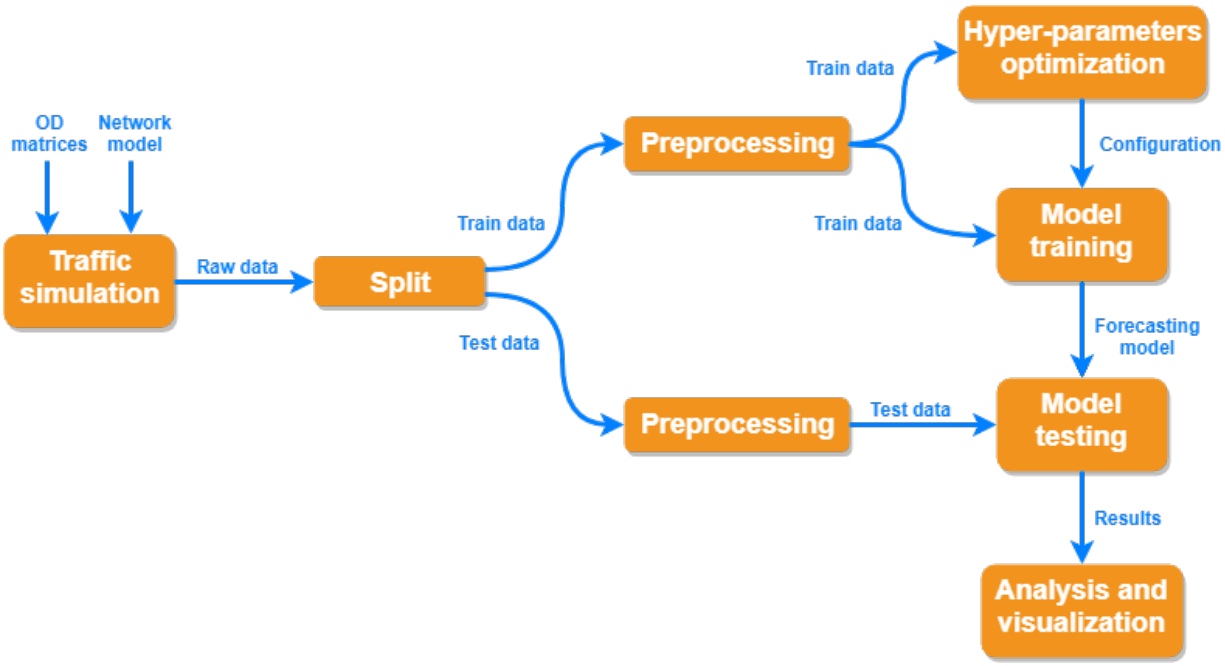


Figure 4.9: Simplified schema of the data flow between the different developed parts of the project.

Model description

The HGC-LSTM model, like the SRCN model, is a combination of CNN layers (described in Section 3.3) and LSTM layers (described in section 3.2.1). The main structure of the model is the same that in the SRCN model case (Figure 4.6). It is formed by one or more CNN layers, a flatten one, the LSTM layers and the final fully connected layer. But, in practice, the two models do not have to be the same given that the final configuration is determined in the training process (included in Section 5.3). For example, the final SRCN model can be formed by many CNN layers and a unique LSTM layer, and the HGC-LSTM model can be formed by a unique CNN and many LSTM layers. In fact, given that the input data format is different, probably the structure will be different because the models have to extract the information in different ways.

4.2 Implementation

This Section describes the implementation of the whole project, as well as the tools used. The complete process includes from the data generation until the visualization of the final results. Figure 4.9 shows a simplified schema of the data flow between the different developed parts. It is simplified because some models require extra processes that are not included. In the following Subsections, the complete data flow is described.

With respect to the tools, aside from the traffic simulation part, all the components of the project

have been developed in *Python 3.6*[1]. Also, diverse *Python* libraries are included in this software in a transversal way in order to work with the data and perform some operations over it. The most important are *NumPy*[2], *Pandas*[3] and *Scikit Learn*[4]. In addition, some other software has been used for specific parts and they will be introduced in the following Subsections.

Because the whole project has been developed with *Python*, the *conda*[5] environment manager is used to build the development environment and keep it controlled and replicable in an easy way. In order to write the code in a comfortable way and get some results during the execution, the *Jupyter Notebook*[6] framework has been used. This is an open-source web application that allows to program *Python* scripts and generates reports in a dynamic way. Finally, all the code is included in a *Git*[7] repository in *GitLab*[8].

4.2.1 Traffic simulation

The first step is to generate the raw data, which is the FCD produced by the microscopic traffic simulator named *Aimsun*[9]. As it is explained in Section 5.1, in order to perform a traffic simulation, the software requires some inputs like to OD matrices and the network model. The main feature of a microscopic simulator is that they perform the complete simulation by emulating the behavior of each vehicle. To do this, the execution is divided into short cycles where the next simulation step is decided. *Aimsun* allows to include custom operations in these cycles integrating *Python* or *C++* scripts. In total, three *Python* scripts have been developed to extract information from the simulation:

- *DataGathering:* The developed *Python* script is executed every predefined time, skipping the cycles between it. Each execution, the script checks all the vehicles in the network and for each one, it saves the vehicle identifier, the timestamp, the current speed, the road section, and the road lane. These records are saved in a csv file and it will be the input FCD for the rest of the process.
- *NetworkMapping:* This script is executed when the simulation starts and defines a mapping from the network roads to the pixels of a figured image. The process checks all the sections of the network and builds a global bounding box that contains all of them. This global box is divided into a grid of a defined number of sub-boxes, which represents the sub-regions of the whole urban scenario. So, the script checks for each network sections in what sub-boxes it is and assigns the section to the corresponding sub-boxes. Finally, a list of all sub-regions with the list of assigned sections for each one is saved in a csv file.
- *SectionsInfo:* Sections adjacency and properties: Like the previous script, this one is executed at the beginning of the simulation. In this case, the process saves two csv files with the information of

[1] `https://www.python.org/`
[2] `https://www.numpy.org/`
[3] `https://pandas.pydata.org/`
[4] `https://scikit-learn.org/`
[5] `https://conda.io/`
[6] `https://jupyter.org/`
[7] `https://git-scm.com/`
[8] `https://gitlab.com/`
[9] `https://www.aimsun.com/`

the section. The first one contains the adjacency between the network sections. Two sections are adjacent if it exists some turning to change from one to another. The second file saves the basic information for all the road sections: section identifier, length, and maximum speed.

In this part of the process, the main tool used is the *Aimsun* simulator. Concretely, the version 8.3.0 of the *Aimsun Next* software with a professional license. The scripts are developed in *Python 2*.

4.2.2 Split and preprocessing

At the end of the traffic simulation, the FCD is split into train and test subsets. The main goal of this division is to train the forecasting model with different data used to test it. This process selects the data corresponding to a number of days and assigns it to the train set. The rest of the data corresponding to the later days is assigned to the test set.

Once the data has been separated, the two subsets are preprocessed to the required data for the forecasting models. The basic preprocessing is described in detail in Section 4.1.1. But summarising, this process aggregates the raw data by intervals, imputes the missing values, scales it and transforms it in supervised data defining an input and an output dataset. For the SRCN and HGC-LSTM models, the generation of the input data requires some extra operations that use information from the network. This information is extracted by *NetworkMapping* and *SectionsInfo* scripts.

In the SRCN model case, the input is composed by a set of images that have to be generated. For this, the collected FCD and the file generated with the simulator script named *NetworkMapping* are used. This file contains the mapping of the network road to an image and the transformation is described in Section 4.1.2.

For the HGC-LSTM model, the input is a numeric matrix computed from the FCD, the adjacency matrix of the network and the FFR matrix. These matrices and the process to join them are described in Section 4.1.3. The adjacency matrix is a typical structure in the graph theory that defines the feasibility to go from a node (a section in this case) to another in a single step. The FFR matrix (Free-Flow Reachability matrix) defines the feasibility of a vehicle to go from a section to another in a determined time. To build this matrix, the shortest path between the networks section is computed using the *Dijkstra* algorithm and the costs are the free flow travel times computed using the length and the maximum speed of the sections in the shortest paths. Both matrices are built with the *sections adjacency and properties* data extracted with the simulator scripts.

For these two components, two tools have been added to the technological stack. The *Python* package *fancyimpute*[10] that implements different techniques to impute missing values in a dataset. And, *imageio*[11], which is another *Python* package used to work with images, in this case, to generate the input for the SRCN model.

[10] `https://pypi.org/project/fancyimpute/`
[11] `https://pypi.org/project/imageio/`

4.2.3 Hyper-parameters optimization and model training

The hyper-parameters optimization consists in finding a good configuration (ideally, the best) to build the best possible forecasting model for the problem that it is used. This process is described in Section 5.3. The used algorithm for the hyper-parameters optimization requires to build and test many models with different configurations. To do this, the training data is split into two subsets named train and validation. The mechanism is the same that in the split step, and the goal is to test the models with data that is not used in the train of the model. The hyper-parameters optimization finishes with the best configuration found in function to their accuracy in the validation data. This best configuration is used in the training of the final forecasting model. Once the configuration for the model has been decided, the model is built using it and it is trained with the whole train data.

In order to work with deep learning models, the *Keras*[12] API has been used. It is a high-level neural networks API, written in *Python*. Although it is capable of running on top of different backends, in this project the *TensorFlow*[13] option is used. This is a machine learning library which implements many methods in an easy usage way but offering a low-level control. The main feature for the use of *TensorFlow* is that it is able to improve the efficiency of the performed models through its compiler that increases the speed of the linear algebra operations and allows to use GPU's instead of CPU's if it is available.

4.2.4 Model testing, results analysis and visualization

Finally, the forecasting model performed in the previous part has to be tested in order to evaluate its accuracy. To do this, the test data separated before will be used. The process consists of using the model to predict the outputs for the test data and compare the predictions with the real outputs. To measure accuracy, the error measures defined in Section 5.2 are used. Lastly, these measurements are analyzed with the accuracy of the other performed models and are visualized using tables and plots. The plots are performed using the *Matplotlib*[14] *Python* package.

[12] `https://keras.io/`
[13] `https://www.tensorflow.org/`
[14] `https://matplotlib.org/`

Chapter 5

Computational experiments

This Chapter presents the performed computational experiments and their results. Previously, the data generation process is described. Thus, the error measures used to evaluate the accuracy of the forecasting models are specified. In addition, the method selected for the hyper-parameters optimization is presented together with its configuration.

5.1 Simulated data

This Section describes the data used in this research project. As it is advanced in the state of the art conclusions (Section 2.5), the proposal is to develop different methods to perform traffic forecasting in urban scenarios using Floating Car Data (FCD). In this study, it is proposed to use a traffic simulation software, concretely *Aimsun*, to generate the FCD. Contrary to use real FCD, the generation of data by simulation allows creating data for many different scenarios in order to study the behavior of the models under different situations. In addition, this saves a lot of effort in terms of time and cost respect to the collection of real data. That said, depending on the case, it is better to use real or simulated data. If the goal is the use of the traffic forecasting model in a real scenario, real data is highly recommended. Else, if the goal is to compare different methods and evaluate their performing under different conditions in a more general way, the simulated data is a better option.

A traffic simulator is a software able to reproduce the complex traffic flow dynamics in order to develop meaningful operational strategies for real-time situations. There are three types of traffic simulation, which are distinguished by the level of granularity that represents the studied system:

- **Macroscopic simulation:** macroscopic simulators treat traffic in an aggregate manner, such as a uniform or homogeneous flow, without considering each constituent particle (individual vehicle). They approximate flow propagation throughout the network using physical concepts or analytical

methods.

- **Microscopic simulation:** microscopic approaches model individual entities, decisions, and interactions with a higher degree of detail. Each vehicle maneuvers at a specific simulation time step based on estimations derived from a set of behavioral models, such as car following, lane changing, merging and yielding.
- **Mesoscopic simulation:** mesoscopic simulators combine elements from microscopic and macroscopic approaches, representing the activities and interactions of each vehicle with less detail, but still enough to account for the essentials of traffic dynamics.

Because the nature of the FCD required, the traffic simulator used has to be a microscopic simulator. This kind of simulator allows to simulate the interactions for each vehicle and so, collect data for each vehicle. Given the project requirements, the previous expertise of the team in the software and the availability of the software license in the inLab, the simulator used is *Aimsun* (Advanced Integrated Microscopic Simulation and Urban Networks). *Aimsun* allows to perform microscopic simulations and execute custom scripts in each iteration in order to collect data.

The traffic simulators require some inputs to perform a simulation. These inputs are the demand, that has to be loaded into the network, and the road network characteristics. The following Subsections describe the inputs used in the project, splitting them into traffic demand and traffic networks.

5.1.1 Traffic demand

The traffic demand is the number of travels that have to be performed in a simulation as well as their origins, their destinations and their departure time. Depending on the available data, instead of setting a specific departure time, it is used a departure time interval. So, the demand is divided into discrete departure time periods and is represented by a set of time-dependent origin-destination matrices (OD matrices) that contains one matrix for each departure time interval. Each matrix contains the number of trips from each origin to each destination of the network at the corresponding departure time interval.

The networks used in the project only include an OD matrix for a departure time interval of one hour (from 8:00h to 9:00h). In order to perform traffic forecasting with data from every hour of the day, the 23 remaining daily OD matrices needed are extrapolated from the available one. Given the differences between the urban traffic for the different weekdays, the project is focused in making traffic predictions using data of workdays (Monday, Tuesday, Wednesday, Thursday and Friday) and only for these days. So, to extrapolate the available data for the rest of the day, the daily trips profile published in the *Daily mobility survey of Catalunya in 2006*[1] is supposed. To do this extrapolation, the remainder matrices are computed multiplying the available one by a factor. This document does not include the needed factors directly, but it can be deduced from the number of trips (Figure 5.1) for all the transportation modes and the distribution of the hourly trips between the different transportation modes (Figure 5.2).

[1] `https://www.atm.cat/web/en/observatori/mobility-surveys.php`

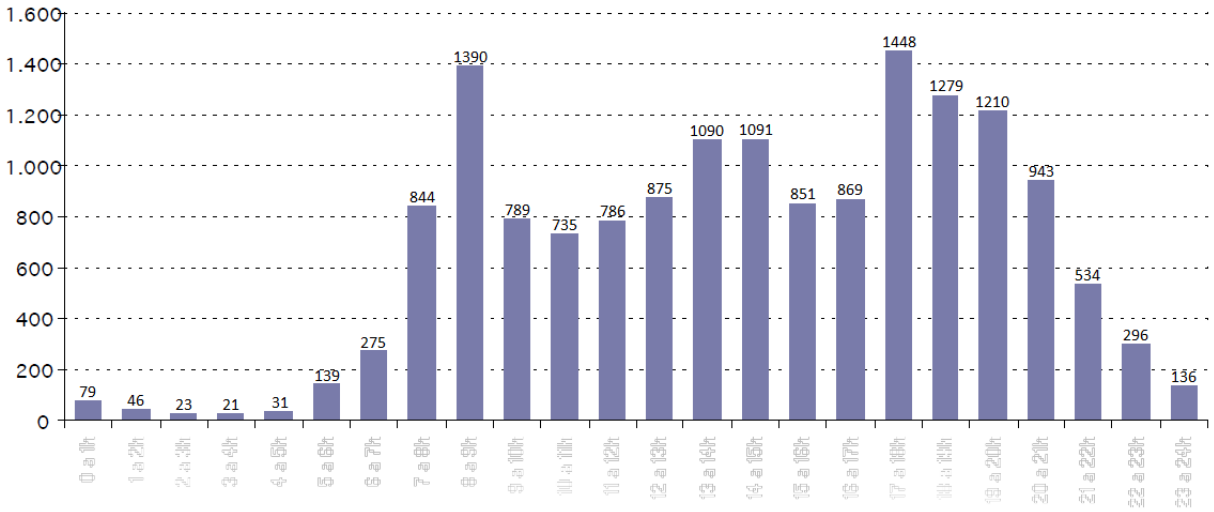


Figure 5.1: Distribution of the number of daily trips (in thousands) in a working day. *Source:* `https://www.atm.cat/web/en/observatori/mobility-surveys.php`

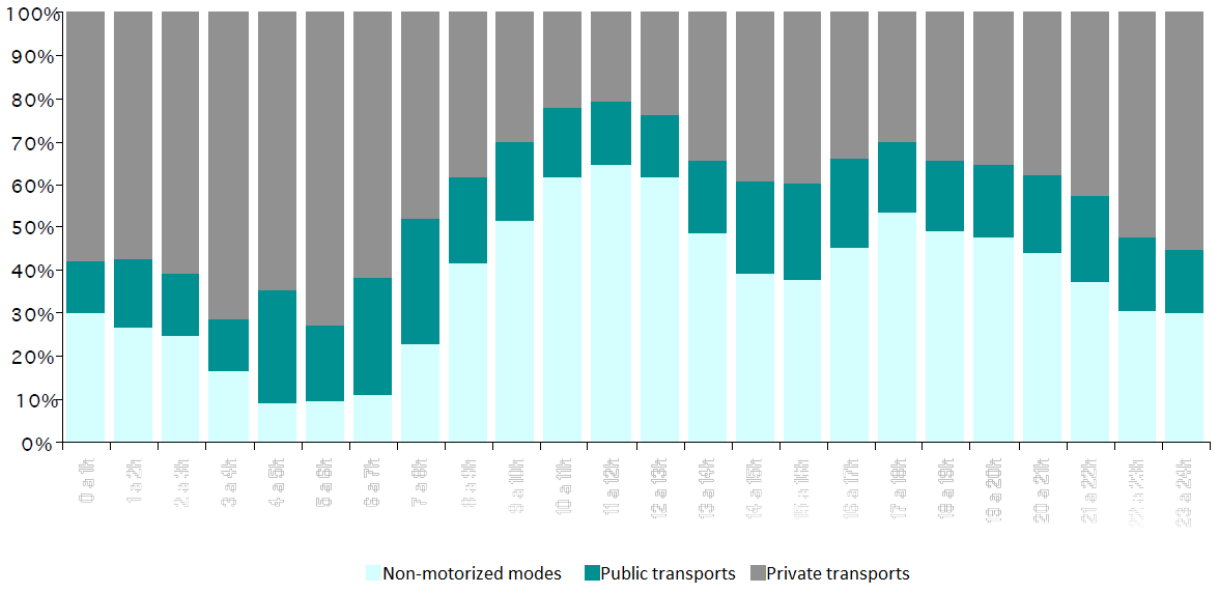


Figure 5.2: Distribution of trips in function of the transportation mode in a working day. *Source:* `https://www.atm.cat/web/en/observatori/mobility-surveys.php`

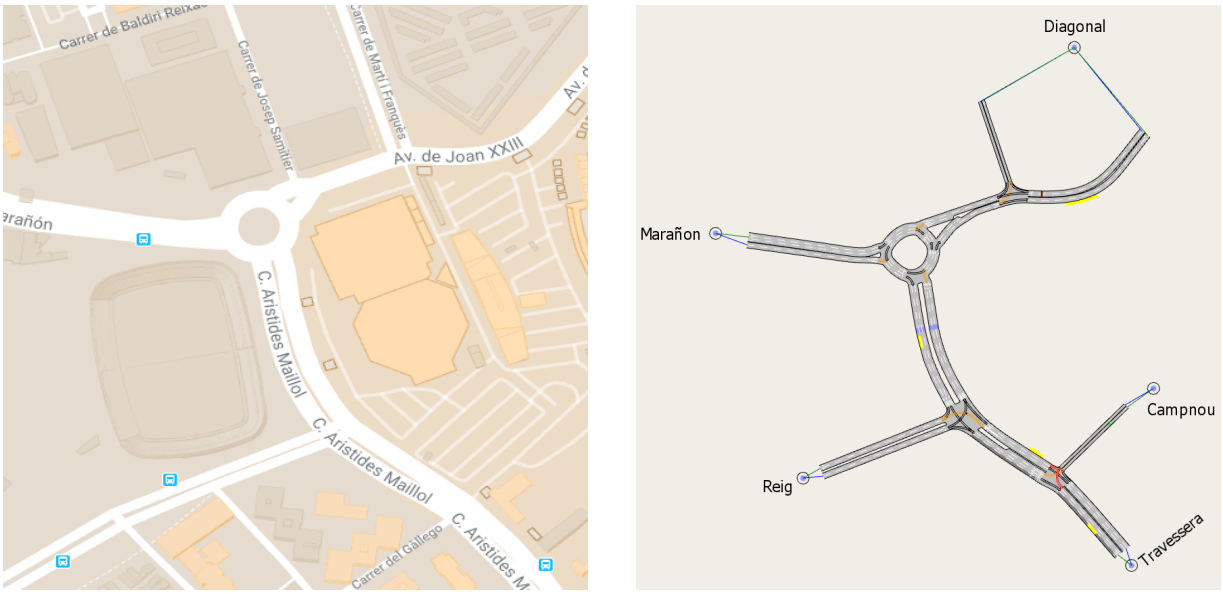


Figure 5.3: Left, the map of the real simulated zone in the Camp Nou model *Source:* `https://www.google.com/maps/`. Right, the Camp Nou simulation model schema.

5.1.2 Traffic networks

In order to study the behavior of the traffic forecasting models with different types of networks, some traffic networks are used. This is important to detect if the developed methods work better under some network features like the size, the capacity, the topology of the roads, etc. The following Subsections describe the different traffic networks used in this project.

Camp Nou

The urban network represented in this model is located in the city of Barcelona, in Spain. Concretely, the network is a little city area composed of 4 nodes and 22 sections. In total, the network represents $1.96km$ of road length. The maximum speed of all road sections is $50km/h$. In Figure 5.3 the map of the real zone and the simulation model schema are shown. Although this is a very little network, it is common in the literature to test the forecasting traffic model with networks of this size. It is very useful to analyze in more detail the behavior of the prediction model to detect possible errors.

Amara

The urban network represented in this model is located in the city of San Sebastián, in Spain. Concretely, the network is a district of the city composed of 105 nodes and 192 sections. In total, the network represents $15.79km$ of road length. The maximum speeds of the road sections are between $35km/h$ and $60km/h$. In Figure 5.4 the map of the real zone and the simulation model schema are shown. In contrast

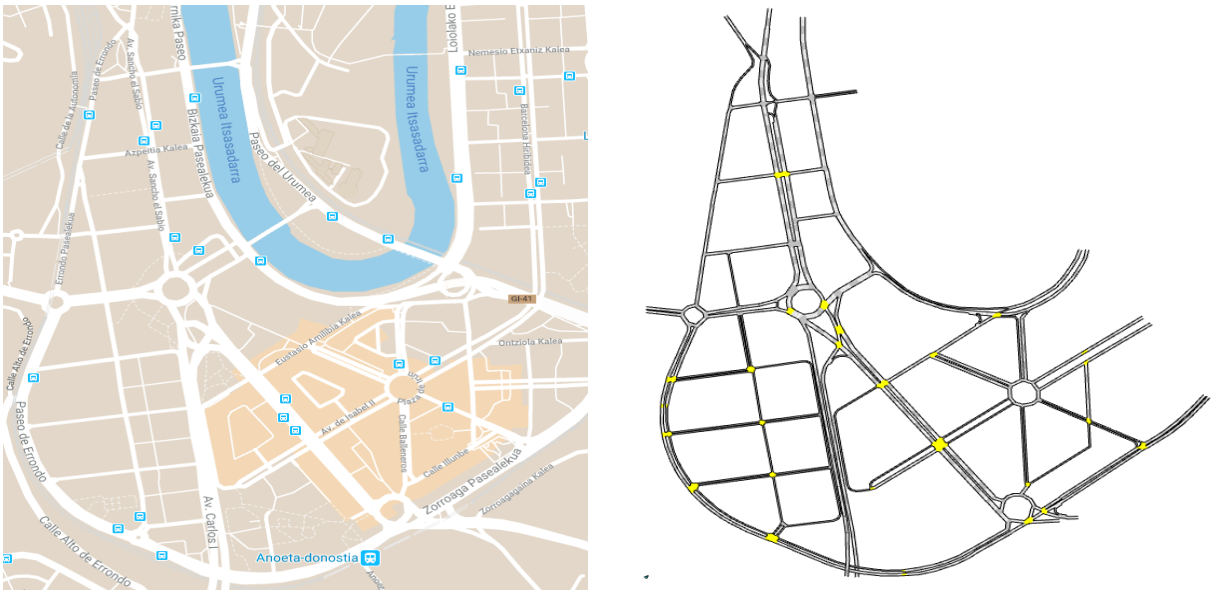


Figure 5.4: Left, the map of the real simulated zone in the Amara model *Source:* `https://www.google.com/maps/`. Right, the Amara simulation model schema.

to the Camp Nou model, this network is bigger and more complete. It is interesting to test the forecasting models in a realistic network like this, despite the size of the network hinder the analysis of the performed prediction in depth.

5.2 Error measures

In order to evaluate the performance of a forecasting model, the results forecast error is used. A forecast error quantifies how well the forecasted values $\widetilde{y} \in \mathbb{R}^N$ match the observed ones $y \in \mathbb{R}^N$. The forecast accuracy can be measured by summarising the forecast errors in different ways called error measures. In the regression setting, the two most commonly used measures are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE).

The MAE is the most simple measure and easy to interpret. When comparing forecast methods applied to a single continuous variable, or to several continuous variables with the same units (km/h in this case), the MAE is popular as it is easy to both understand and compute. It simply computes the average of the absolute differences between the observed and the predicted values (Equation 5.1).

$$MAE = \frac{1}{N} \sum_{i}^{N} \mid y_i - \widetilde{y}_i \mid \tag{5.1}$$

Probably, the most commonly used measure in regression problems is the RMSE. Despite it is more difficult to interpret because it penalizes much more the big errors than the small ones, the units are also

the same (km/h in this case). So, the models performed by minimizing the RMSE should be more useful when large errors are particularly undesirable. The main difference with the MAE is that the RMSE does not use the absolute difference but the squared difference. Also, the final mean error is rooted as Equation 5.2 shows.

$$RMSE = \sqrt{\sum_{i}^{N}(y_i - \widetilde{y}_i)^2} \tag{5.2}$$

5.3 Hyper-parameters optimization

Most of the machine learning methods has to be configured before to be used. This requires to set a group of parameters (dependent on each method) with the optimal values in order to achieve the best performance of the model. This process is named hyper-parameter optimization (or tuning) and it consists in try many different values for the parameters and select the configuration that produces the best model performance following some criteria, for example, the accuracy or the computational time.

5.3.1 Hyper-parameter optimization algorithm

The tuning can be aboard as an optimization problem where the parameter values are optimized in function of the desired criteria. The problem of this is that each iteration of the optimization execution requires to train and test a model to compute the objective function, which can be a computational costly task. Because this, frequently the search of the optimal solution is not feasible using an exact solver and a trial-error strategy is used to search for good configurations. So, the critical step in hyper-parameter optimization is to choose the set of trials in order to find for the best configuration (or a good enough one).

Many different strategies exist in order to choose the set of configurations to test. The most widely used strategy is a combination of the grid search and the manual search. In the grid search, the set of trials is formed by assembling every possible combination of values. Except for the numeric parameters, which requires a previous selection of the possible values for each parameter. Manual search is used to identify search regions that are promising and to develop the intuition necessary to choose the trial set. This kind of search is totally dependent on the user criteria and it is difficult to reproduce and automatize the experiments.

Bergstra and Yoshua [2012] proposes random search as alternative of grid search and manual search. The random search consists in generating the set of trials randomly. The results of the paper show that this strategy is able to find models that are as good or better within a small fraction of the computation time. Also, it keeps the advantages of simple, reproducible and automatable implementation of the grid search. Thus, the random search is used in this project for the hyper-parameters optimization of the

developed methods.

About the needed number of iterations, there is a simple probabilistic explanation about the result: for any distribution over a sample space with a finite maximum, 60 random observations contains at least one of the top 5% instances with 95% probability. Each random observation has a 5% probability to be contained in the 5% top, and its probability of being out of this group is $1 - 0.05$. For n independent observations, the probability of all of them are out of the top 5% is $(1 - 0.05)^n$, and $1 - (1 - 0.05)^n$ is the probability of at least one observation is inside the 5% top. So, the n number of needed iterations to get a configuration in 5% top with 95% probability is deduced from the equation 5.3. This equation is solved with $n \geq 60$. So, to perform the hyper-parameter optimization of the models of this project, the random search algorithm will be used with 60 iterations for each tuned model.

$$1 - (1 - 0.05)^n \geq 0.95 \tag{5.3}$$

5.3.2 Hyper-parameter optimization configuration

Besides the hyper-parameter optimization algorithm, this process requires taking some basic configurations that can affect the results. One of the most important ones is what data will be used and how. The used raw data is generated as it is explained in Section 5.1. Also, the input data of each method is preprocessed as it is explained in the *Input data preprocessing* Sections of Chapter 4.1.

In order to evaluate the different trials tested in the tuning process, the 20% of the data is used as validation data. So, this percentage of the data is not used in the training and only is used to evaluate the model. It is the equivalent to the test data for the tuning process. The quantity of data used for tuning is the data generated for a period of 5 days, but it can change depending on the experiment. The measure to evaluate the model performance is the *Mean Squared Error*.

The neural networks methods (including deep learning ones) requires to define two hyper-parameters to configure the use of the data during the training. This process is an iterative algorithm used in machine learning to optimize the weights of the model. So, in each iteration, the model is trained with more data and the internal parameters are updated. In this process, the following hyper-parameters are determinant:

- **Batch size**: The number of train samples per iteration.
- **Epochs**: The number of epochs to train the model. An epoch is a pass over the entire train dataset.

In all the hyper-parameters optimization performed in this project, these two hyper-parameters are fixed. The batch size used is 128 in order to find a trade-off between the completeness and the required training time. The number of epochs determines how many times the same data is used to train the model. Typically, in the beginning, as the number of epochs grows, the model improves its performance. But, with a big number of epochs (depending on the data and the model), the model overfits the training

data and the accuracy of the model (evaluated with the validation data) decreases. Usually, the number of epochs is decided manually once the model is tunned. But, for the hyper-parameters optimization, it is not practical. Thus, in order to automatize the selection of the number of epochs and to save computational time during the process, the early stopping strategy is applied. This strategy consists in stop automatically the training of the model when the performance of the model starts to decreases. Thus, although the number of epochs is fixed to 2000, in practice, the model never performs 2000 epochs. This limit is established based on the empirical findings during the computational tests. To avoid an early stopping with a fluctuation of the model performance, the strategy is configured with a parameter called patience, which is the number of epochs with no improvement after which training will be stopped. In this case, the patience is fixed in 5 based on the experimental criteria.

5.3.3 Optimized hyper-parameters

The hyper-parameters optimization is common for all traffic forecasting models. But, as it is explained in Section 3.4, each neural networks method has different hyper-parameters to configure and optimize. Thus, the hyper-parameters to be optimized for each traffic forecasting model depends on the neural networks methods they use. This defines two groups of predictive methods. The first one is formed by LSTM and GRU models, which are very similar and have the same hyper-parameters. The second one is formed by the SRCN and HGC-LSTM models, which share the same structure and so, the same hyper-parameters.

LSTM and GRU hyper-parameters

The following list describes all the optimized hyper-parameters for the performed LSTM and GRU models:

- **Units**: It is the size of the cell state of a LSTM layer. In this project, the number of units is an integer in the range from 1 to 100 (both included).
- **Layers**: It is the number of hidden LSTM layers. In this project, the number of layers is an integer in the range from 1 to 10 (both included).
- **Activation function**: It is the function that uses a neuron to compute the output. In this project the different activation functions tested are: tanh (Hyperbolic tangent activation function), relu (Rectified Linear Unit), sigmoid (Sigmoid activation function) and selu (Scaled Exponential Linear Unit).
- **Initializer method**: It is the method used to set the initial values for the intern parameters of the model. In this project, the different initializer methods tested are: he_normal (He normal initializer), lecun_normal (LeCun normal initializer), glorot_normal (Glorot normal initializer) and random_normal (Random normal initializer).
- **Dropout**: It is the percentage of units randomly selected to be excluded from activation and weights update while training a network. In this project, the dropout is a real number in the range from 0.0

to 0.6.

- **Optimizer**: It is the optimization algorithm used in the training process. In this project the different optimizers tested are: Stochastic gradient descent optimizer (SGD), RMSProp optimizer (RMSProp), Adagrad optimizer (Adagrad), Adadelta optimizer (Adadelta), Adam optimizer (Adam), Adamax optimizer (Adamax) and Nesterov Adam optimizer (Nadam).

SRCN and HGC-LSTM hyper-parameters

Because the structure of the SRCN and HGC-LSTM methods includes a set of LSTM (or GRU) layers, the hyper-parameters of the LSTM models are also hyper-parameters of these methods. The following list describes the optimized hyper-parameters for the performed SRCN and HGC-LSTM models, except those included in the previous Section:

- **RNN method**: It is the type of RNN layer used. Although the original methods use LSTM layers, in this project the GRU is considered as an alternative to LSTM in traffic forecasting. Also in these methods, the GRU layers are considered as an alternative of LSTM layers. So, the options are: LSTM and GRU.
- **CNN filters**: It is the number of filters for each convolutional layer. In this project, the number of filters is a power of two of the range from 2 to 128 (both included).
- **CNN filter window size**: It is the size of the windows of the image where the filters are applied. In this project only are used square filters, so the size is an integer in the range from 2 to 10 (both included) considered the height and the width of the window.
- **CNN activation function**: It is the activation function used in the CNN layers, which has not to be the same that the activation function used in the LSTM layers. The options are the same that in the LSTM layers case.
- **CNN initializer method**: It is the initializer method used in the CNN layers, which has not to be the same that the initializer method used in the LSTM layers. The options are the same that in the LSTM layers case.
- **CNN pooling window size**: It is the size of the windows of the image where the pooling is applied. In this project the pooling is used in square regions, so the size is an integer in the range from 2 to 10 (both included) considered the height and the width of the window.
- **CNN pooling function**: It is the function that uses a pooling layer to combine the elements of a submatrix. In this project, the options considered are: Max (Max pooling operation) and Avg (Average pooling operation).
- **CNN layers**: It is the number of hidden CNN layers. In this project, the number of layers is an integer in the range from 1 to 10 (both included).

5.4 Experiments

This Section presents the different experiments performed in this master thesis and their results. The main goal of these experiments is to evaluate how works the different proposed models in different traffic forecasting contexts. All the experiments have an identification name that codifies the context that is testing. For example, the name of the experiment *EX-LSTM-CN-PR100* is composed of four shortcodes:

- *EX:* The abbreviation of the word *Experiment*. It is a prefix for all the experiments.
- *LSTM:* It is the acronym of the used forecasting method. The possible values are *LSTM* for Long Short-Term Memory, *GRU* for Gated Recurrent Unit Neural Networks, *SRCN* for Spatiotemporal Recurrent Convolutional Networks and *HGC* for High-Order Graph Convolutional Long Short-Term Memory Neural Network.
- *CN:* It is the acronym of the urban network scenario used. The possible values are *CN* for Camp Nou and *AM* for Amara.
- *PR100:* The letters refer to the feature tested (Penetration Rate) and the number is the value set (100%). It is specified for each experimented feature.

In addition, some other aspects have been fixed to perform all the experiments, except for the experiments that test different values of these parameters. The data used is the data collected every 10 second in a simulation of 7 days with a penetration rate of 100%. This data is grouped in time periods of 5 minutes and it is split into 3 datasets: train (first 4 days), validation (the next day), and test (with the 2 last days). The presented results is an evaluation of the predictions in a horizon of 5 minutes. For the SRCN model, the K used is 3. And, for the HGC-LSTM model, the inputs used are images of $32x32$ pixels.

The hyper-parameters of the models developed for these experiments are optimized following the algorithm explained in Section 5.3. As in the case of the hyper-parameters optimization, the models need two additional parameters: batch size and number of epochs. The values set for these parameters are the same that in Section 5.3 (batch size 128 and the number of epochs 2000). Also, the same early stopping strategy is applied with 5 epochs of patience.

All the experiments are executed over the same hardware. It is a computer with a Windows 10 operating system of 64 bits and the following components:

- A CPU *Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz.*
- A hard disk *Samsung SSD 850 EVO M.2 500GB.*
- 4 RAM modules of 16GB each (64GB in total).
- A GPU *NVIDIA GeForce GTX TITAN X.*

The metrics used for the evaluation of the experiments are the two error measures described in Section 5.2 (RMSE and MAE) and the model training computational time in seconds. In the following subsections, the experiments are described and their results are detailed.

5.4.1 FCD penetration rate

Currently, more and more vehicles incorporate GPS and similar systems to offers better services to the user. Although this is the main way to generate floating car data, it is not the only one. Also, most people bring connected smartphones with them, which is also a good source to collect FCD. Despite this, not all the vehicles in an urban network can be tracked through some of these systems. The percentage of vehicles that can be tracked is named the penetration rate.

When the used Floating Car Data is real data, the penetration rate has a strong impact on the results. Actually, it is very difficult to imagine an urban network with a penetration rate of 100%. So, it is very interesting to evaluate how works a traffic forecasting model with different penetration rates in order to study its feasibility in a real scenario. The simulated data allows testing this.

Experiment design

Table 5.1 summarizes the proposed experiments and details the corresponding levels of the design factors. The experiments combine the four different prediction models, the four different penetration rates (100%, 75%, 50% and 25%) and the two proposed scenarios (Camp Nou and Amara).

In order to perform the experiments with different penetration rates pr, an extra preprocessing step is added. This consists in select randomly a set of vehicles from the raw data and delete their records before to groups the data. The length of this set is the total number of vehicles of the data multiplied by $1 - pr$.

In addition to the standard error measures used for all the experiments, the penetration rate affects directly to the number of missing values of the dataset, because records removing increase the probability of time periods without data for the road sections. Consequently, the number of missing values is shown in the results sections in function of the penetration rates.

Camp Nou network results

In this Subection, the results of the proposed experiments executed over the Camp Nou network (described in Section 5.1.2) are analyzed. These results are shown in Table 5.2. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.5 shows a plot for each error measure. These plots contain the error for each model and for each proposed penetration rate.

As it is expected, the error of all the models decreases when the penetration rate grows. In this scenario, the accuracy of the four models it is very similar. In the ideal situation (penetration rate of

Experiment	Model	Scenario	Penetration rate
EX-LSTM-CN-PR25	LSTM	Camp Nou	25%
EX-LSTM-CN-PR50			50%
EX-LSTM-CN-PR75			75%
EX-LSTM-CN-PR100			100%
EX-LSTM-AM-PR25		Amara	25%
EX-LSTM-AM-PR50			50%
EX-LSTM-AM-PR75			75%
EX-LSTM-AM-PR100			100%
EX-GRU-CN-PR25	GRU	Camp Nou	25%
EX-GRU-CN-PR50			50%
EX-GRU-CN-PR75			75%
EX-GRU-CN-PR100			100%
EX-GRU-AM-PR25		Amara	25%
EX-GRU-AM-PR50			50%
EX-GRU-AM-PR75			75%
EX-GRU-AM-PR100			100%
EX-SRCN-CN-PR25	SRCN	Camp Nou	25%
EX-SRCN-CN-PR50			50%
EX-SRCN-CN-PR75			75%
EX-SRCN-CN-PR100			100%
EX-SRCN-AM-PR25		Amara	25%
EX-SRCN-AM-PR50			50%
EX-SRCN-AM-PR75			75%
EX-SRCN-AM-PR100			100%
EX-HGC-CN-PR25	HGC-LSTM	Camp Nou	25%
EX-HGC-CN-PR50			50%
EX-HGC-CN-PR75			75%
EX-HGC-CN-PR100			100%
EX-HGC-AM-PR25		Amara	25%
EX-HGC-AM-PR50			50%
EX-HGC-AM-PR75			75%
EX-HGC-AM-PR100			100%

Table 5.1: Set of the proposed experiments for the penetration rate.

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-CN-PR25	4.72	8.227	6.8
EX-LSTM-CN-PR50	3.963	6.134	72.6
EX-LSTM-CN-PR75	3.128	5.196	51.9
EX-LSTM-CN-PR100	2.932	4.865	22.6
EX-GRU-CN-PR25	4.837	8.318	11.0
EX-GRU-CN-PR50	3.677	5.817	24.6
EX-GRU-CN-PR75	3.287	5.292	71.3
EX-GRU-CN-PR100	2.862	4.776	52.2
EX-SRCN-CN-PR25	4.857	8.27	21.8
EX-SRCN-CN-PR50	3.932	5.937	12.8
EX-SRCN-CN-PR75	3.394	5.436	6.9
EX-SRCN-CN-PR100	3.197	5.12	5.2
EX-HGC-CN-PR25	4.731	8.306	10.5
EX-HGC-CN-PR50	3.637	5.836	7.6
EX-HGC-CN-PR75	3.322	5.408	7.3
EX-HGC-CN-PR100	3.104	5.03	8.8

Table 5.2: Results of the penetration rate experiments in the Camp Nou network.

100%), the LSTM and the GRU models perform the most accurate predictions, with a short advantage for the GRU option. On removing the 25% of the vehicles, the GRU model resembles its error to the SRCN and the HGC-LSTM models, being the LSTM model the best option in terms of error. With less than the 75% of vehicles, the accuracy of the LSTM model decreases rapidly. With the rest of the models, this strong decrease occurs with penetration rate lower than 50%. With the lowest penetration rate (25%) the performance of the models gets much worse.

In terms of training computational time, and as it is shown in Figure 5.6, the penetration rate affects the different models in different ways. The SRCN and the HGC-LSTM models show the smaller training times except with the smaller penetration (25%), when LSTM and GRU are similar to HGC-LSTM and faster than SRCN. In the rest of the penetration rates, the LSTM and GRU are slower than the convolutional options, presenting the worst values for 50% and 75%, respectively.

Moreover, Figure 5.7 shows the result of the percentage of missing values in function of the penetration rate. As it is expected, the number of missing values grows when the penetration rate decreases. In the ideal situation, with all the vehicles connected, the minimum possible percentage of missing values for this network is 9.2%. And it increases more strongly with lower penetration rates. In fact, the difference of the percentage of missing values between the studied penetration rates shows exponential increments (2.793%, 4.636%, and 8.859%).

Amara network results

In this Subection, the results of the proposed experiments executed over the Amara network (described in Section 5.1.2) are analyzed. These results are shown in Table 5.3. To begin with, the results for the

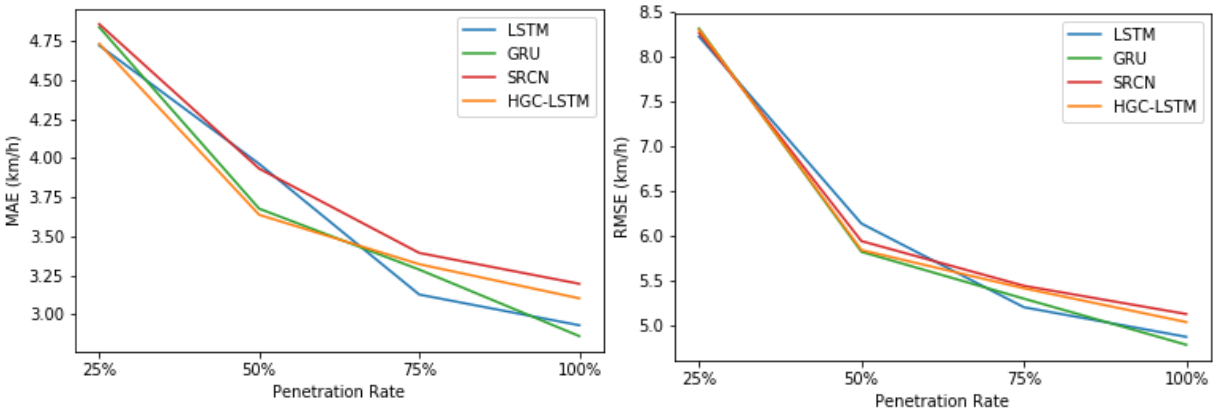


Figure 5.5: Plots of the MAE (left) and RMSE (right) errors of the models in function of the penetration rate in the Camp Nou network.

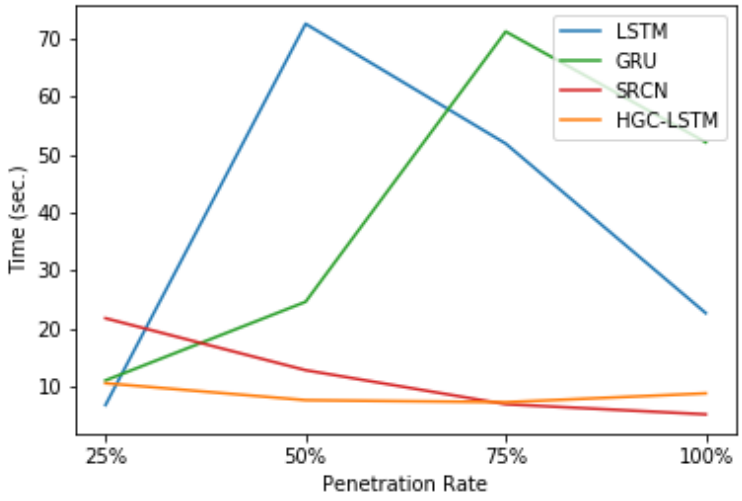


Figure 5.6: Plots of the training computational time of the models in function of the penetration rate in the Camp Nou network.

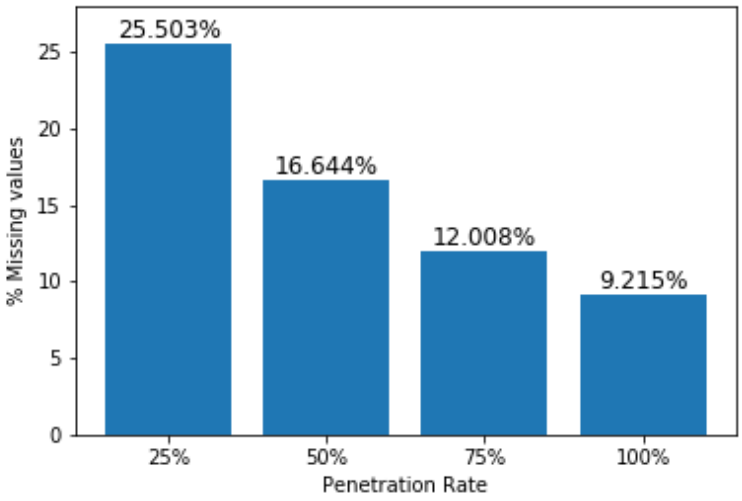


Figure 5.7: Plots of the number of missing values in function of the penetration rate in the Camp Nou network.

MAE and the RMSE measures are presented. Figure 5.8 shows a plot for each error measure. These plots contain the error for each model and for each penetration rate.

Following the expected trend, like to in the Camp Nou network, the error of all the models decreases each time the penetration rate grows in the Amara scenario too. But in this case, the order of the models does not change for all the penetration rates. LSTM and GRU are the best models for all the values with a short advantage for the LSTM option. HGC-LSTM and after SRCN complete the ranking. The LSTM, GRU and SRCN models show a bit improvement decreasing the penetration rate from 75% to 50%. A possible reason could be that given to the hyper-parameters optimization for each experiment, the models change its configurations and performs differently. But the logs of the process shows that the 3 models use the same configuration between the two cases. This is a striking phenomenon and could be interesting to do experiments with intermediate penetration rates to check the variations.

In terms of training computational time, and as it is shown in Figure 5.9, the penetration rate affects the different models in different ways. In a similar way to in the Camp Nou case, the convolutional models are the most constants. On another hand, the LSTM model shows an improvement of the time with high penetration rates. The profile of the training time for the GRU model is the inverse one, the time grows with larger penetration rates. And its time for the full penetration (100%) is almost double than the worst time of the rest of models.

Moreover, Figure 5.10 shows the result of the percentage of missing values in function of the penetration rate. Like with the Camp Nou data, the number of missing values grows when the penetration rate increases. But, the percentage of missing values in the Amara network is in general bigger than in the Camp Nou network. In the ideal situation, with all the vehicles connected, the minimum possible percentage of missing values for this network is 39.8%, which is around for times bigger than in the Camp

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-AM-PR25	2.04	4.948	10.8
EX-LSTM-AM-PR50	1.951	4.834	11.4
EX-LSTM-AM-PR75	1.953	4.835	5.8
EX-LSTM-AM-PR100	1.831	4.602	5.7
EX-GRU-AM-PR25	2.045	5.185	2.9
EX-GRU-AM-PR50	1.991	5.016	2.0
EX-GRU-AM-PR75	2.021	5.062	3.4
EX-GRU-AM-PR100	1.88	4.73	20.6
EX-SRCN-AM-PR25	4.026	6.532	6.1
EX-SRCN-AM-PR50	3.681	6.264	6.0
EX-SRCN-AM-PR75	3.723	6.347	6.1
EX-SRCN-AM-PR100	3.368	5.896	6.0
EX-HGC-AM-PR25	3.525	6.034	5.2
EX-HGC-AM-PR50	3.332	5.91	5.6
EX-HGC-AM-PR75	3.186	5.811	5.2
EX-HGC-AM-PR100	2.905	5.412	4.9

Table 5.3: Results of the penetration rate experiments in the Amara network.

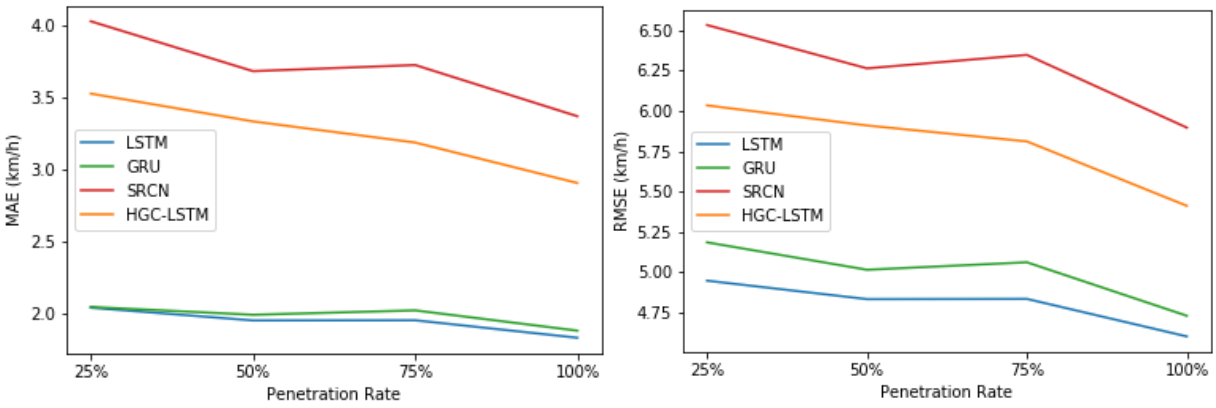


Figure 5.8: Plots of the MAE (left) and RMSE (right) errors of the models in function of the penetration rate in the Amara network.

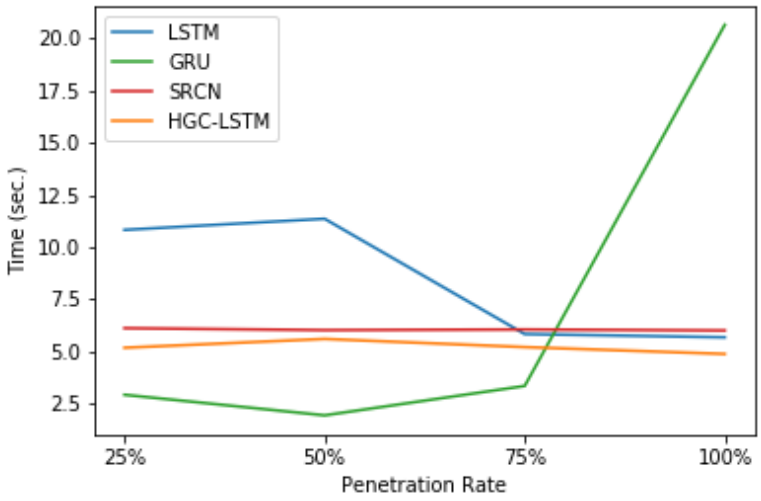


Figure 5.9: Plots of the training computational time of the models in function of the penetration rate in the Amara network.

Nou case. The increment of the number of missing values between the different penetration rates is bigger too (3.912%, 6.004%, 11.291%), but the difference is not so big.

Experiment conclusions

Finally, some conclusions from the previously presented results are included. In terms of forecasting error, the best models for the biggest penetration rates (100% and 75%) are LSTM and GRU. In the big network, these two models are the best also for the lowest penetration rates (50% and 25%). In contrast, for the little network the HGC-LSTM model is the best option for the lowest penetration values.

On the other hand, the decrease of the penetration rate affects directly to the prediction accuracy, which also decrease. In the Camp Nou network, the difference between the 100% and 75% penetration rate is not so big. But, with lower levels, the error grows strongly. In contrast, with the Amara scenario, the error is more constant for the different tested ratios. So, it can be concluded that a network with a smaller size is more sensible to the penetration rate than a big urban area. In the first case, a penetration rate smaller than 75% shows a big error in the predictions. While, in the second one, although the best results are obtained with the maximum penetration, the methods are able to perform reasonably good traffic forecasting with low ratios.

Moving on, the training computational time of the methods is also analyzed. The convolutional solutions are more stable in the required time and they shows very good times for all the tested cases. The LSTM and GRU methods are more inestable in this terms, specially with the Camp Nou network. In fact, the times registered in the little network are bigger than with the Amara scenario. This may be caused by a faster convergence (less epochs needed) of the forecasting models for a bigger network.

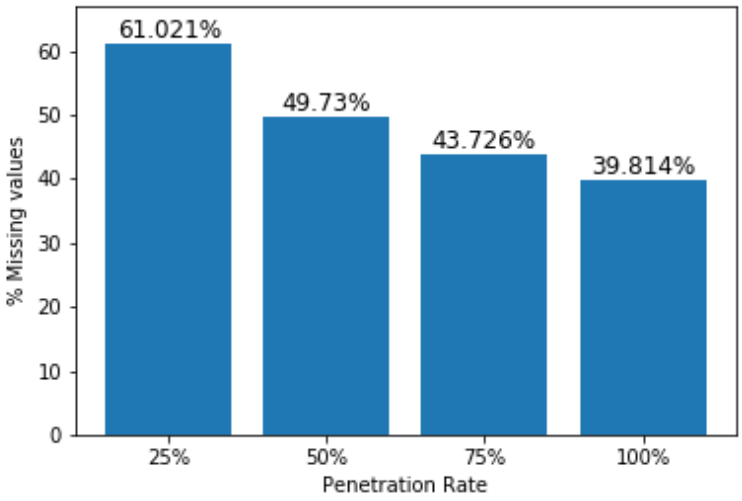


Figure 5.10: Plots of the number of missing values in function of the penetration rate in the Amara network.

Anyway, the worst registered time is slightly bigger than 1 minute, which is a feasible training time for a forecasting model.

In general, and contrary to the intuitively expected, the traffic forecasting errors are lower for the Amara scenario. Another striking result is that the variation of the model errors for different penetration rates in Amara are smaller than in Camp Nou. The most probable explanation for these two aspects is the high number of the missing values seen in the performed experiments. As it is explained in Section 4.1.1, sometimes the input data has no contain any record for a section and this difficults its imputation. These cases are imputed with the maximum allowed speed of the road section, which is a constant value for all the time intervals. So, the prediction of an empty section (a constant value) is trivial for the used forecasting models. The Camp Nou scenario does not present any empty section for any tested penetration rate. But, as it is shown in Figure 5.11, the percentatge of empty sections in the Amara network is significant and it grows with lower penetration rates.

5.4.2 Prediction horizon

One of the main issues in the traffic forecasting problem is the prediction horizon. It defines how far ahead the model predicts the future. As it is explained in Section 1.2, in the traffic forecasting problem the prediction horizon is classified in short-term or long-term. Depending on the traffic forecasting goal, the required prediction horizon changes. For example, for real-time navigation software, the desired forecasting is in short-term in order to modify the indications to the driver and avoid network congestion. Whereas, for a traffic management systems, the desired prediction horizon could be long-term in order to perform some decisions with enough anticipation for implementing them.

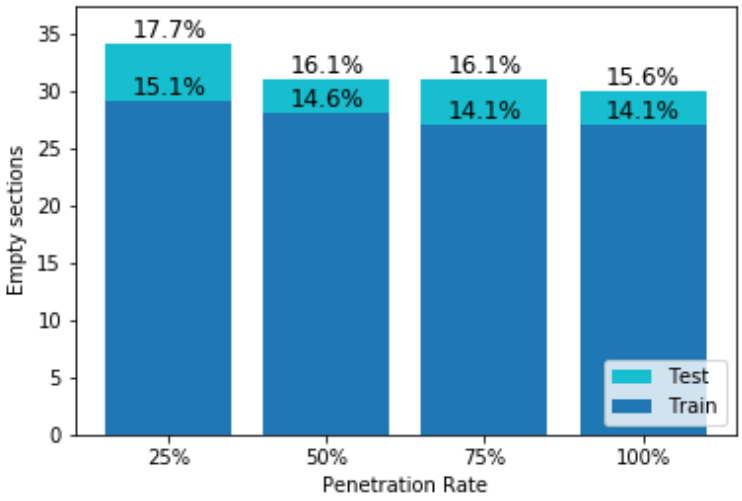


Figure 5.11: Plots of the number of empty sections in function of the penetration rate in the Amara network.

This experiment evaluates how performs each developed model with different short-term and long-term prediction horizons. Also, the importance of the network size is studied.

Experiment design

The prediction horizon experiments are split into short-term, summarized in Table 5.4, and long-term, summarized in Table 5.5. In each table, the configuration of the experiments is included. The experiments combine the four different prediction models, the six different prediction horizons (short-term: 5, 10, 15 and long-term: 20, 40, 60) in minutes and the two proposed scenarios (Camp Nou and Amara).

In order to perform the experiments with different prediction horizons the process of the supervised data generations has been modified. This process takes a parameter that determines the shift between the input data and its expected prediction. For example, when each data instances represents a 5 minutes aggregation (like in the studied case) and the prediction horizon is 5 minutes too, the shift is 1 because the input for an instance i is the aggregation of the next 5 minutes (instance $i+1$). So, to reproduce this experiment, the shift parameter has been reconfigured with the values shown in Table 5.6.

Camp Nou network results for short-term prediction horizons

In this Subection, the results of the proposed experiments executed over the Camp Nou network (described in Section 5.1.2) are analyzed. These results are shown in Table 5.7. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.12 shows a plot for each error measure. These plots contain the error for each model and for each short-term prediction horizon.

Experiment	Model	Scenario	Prediction horizon (min.)
EX-LSTM-CN-PH5	LSTM	Camp Nou	5
EX-LSTM-CN-PH10			10
EX-LSTM-CN-PH15			15
EX-LSTM-AM-PH5		Amara	5
EX-LSTM-AM-PH10			10
EX-LSTM-AM-PH15			15
EX-GRU-CN-PH5	GRU	Camp Nou	5
EX-GRU-CN-PH10			10
EX-GRU-CN-PH15			15
EX-GRU-AM-PH5		Amara	5
EX-GRU-AM-PH10			10
EX-GRU-AM-PH15			15
EX-SRCN-CN-PH5	SRCN	Camp Nou	5
EX-SRCN-CN-PH10			10
EX-SRCN-CN-PH15			15
EX-SRCN-AM-PH5		Amara	5
EX-SRCN-AM-PH10			10
EX-SRCN-AM-PH15			15
EX-HGC-CN-PH5	HGC-LSTM	Camp Nou	5
EX-HGC-CN-PH10			10
EX-HGC-CN-PH15			15
EX-HGC-AM-PH5		Amara	5
EX-HGC-AM-PH10			10
EX-HGC-AM-PH15			15

Table 5.4: Set of the proposed experiments for the short-term prediction horizons.

Experiment	Model	Scenario	Prediction horizon (min.)
EX-LSTM-CN-PH20	LSTM	Camp Nou	20
EX-LSTM-CN-PH40			40
EX-LSTM-CN-PH60			60
EX-LSTM-AM-PH20		Amara	20
EX-LSTM-AM-PH40			40
EX-LSTM-AM-PH60			60
EX-GRU-CN-PH20	GRU	Camp Nou	20
EX-GRU-CN-PH40			40
EX-GRU-CN-PH60			60
EX-GRU-AM-PH20		Amara	20
EX-GRU-AM-PH40			40
EX-GRU-AM-PH60			60
EX-SRCN-CN-PH20	SRCN	Camp Nou	20
EX-SRCN-CN-PH40			40
EX-SRCN-CN-PH60			60
EX-SRCN-AM-PH20		Amara	20
EX-SRCN-AM-PH40			40
EX-SRCN-AM-PH60			60
EX-HGC-CN-PH20	HGC-LSTM	Camp Nou	20
EX-HGC-CN-PH40			40
EX-HGC-CN-PH60			60
EX-HGC-AM-PH20		Amara	20
EX-HGC-AM-PH40			40
EX-HGC-AM-PH60			60

Table 5.5: Set of the proposed experiments for the long-term prediction horizons.

Prediction horizon (min.)	Shift
5	1
10	2
15	3
20	4
40	8
60	12

Table 5.6: Shift value in function of the prediction horizon.

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-CN-PH5	2.932	4.865	22.7
EX-LSTM-CN-PH10	2.917	4.84	23.9
EX-LSTM-CN-PH15	2.88	4.814	6.6
EX-GRU-CN-PH5	2.862	4.776	51.6
EX-GRU-CN-PH10	2.869	4.768	46.2
EX-GRU-CN-PH15	2.873	4.77	40.5
EX-SRCN-CN-PH5	3.197	5.12	5.0
EX-SRCN-CN-PH10	3.181	5.107	5.0
EX-SRCN-CN-PH15	3.18	5.1	5.8
EX-HGC-CN-PH5	3.103	5.03	8.8
EX-HGC-CN-PH10	3.179	5.11	3.8
EX-HGC-CN-PH15	3.322	5.179	8.1

Table 5.7: Results of the short-term prediction horizon experiments in the Camp Nou network.

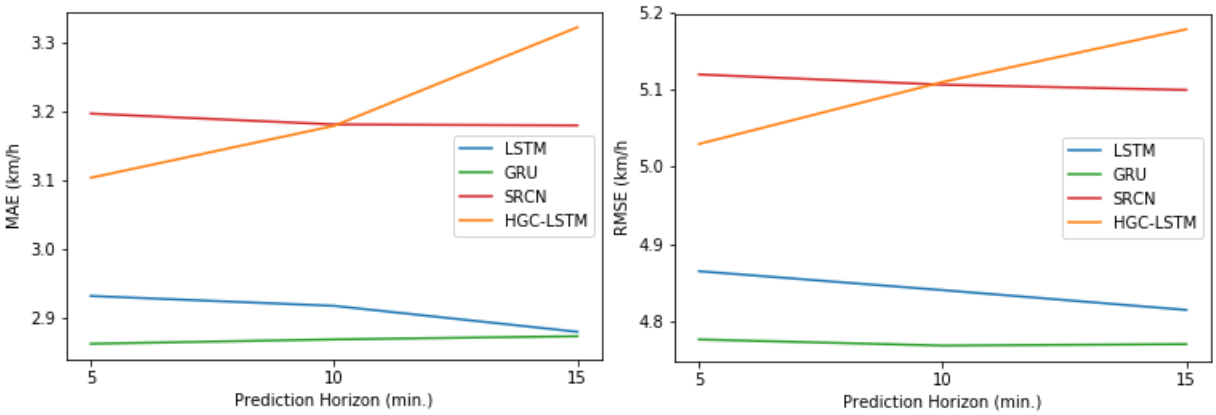


Figure 5.12: Plots of the MAE (left) and RMSE (right) errors of the models in function of the short-term prediction horizon in the Camp Nou network.

In general, the LSTM and GRU models outperform the others models for short-term forecasting in the Camp Nou network. In special, the GRU model shows the best results. The behavior of the GRU and SRCN solutions it is very constant for all the short-term prediction horizon tested. The LSTM model shows a smooth improvement when the prediction horizon grows. In contrast to this, the HCG-LSTM model computes worse predictions with larger prediction horizons. So, this model performs better than SRCN with the smallest prediction horizon (5 minutes), mostly equal with an intermediate prediction horizon (10 minutes) and worse for the largest short-term horizon (15 minutes).

Contrary to the commented error results, in terms of training computational time (shown in Figure 5.13), the fastest models are the SRCN and HCG-LSTM. Concretely, SRCN has an almost constant time of 5 seconds and HGC-LSTM shows worse times than SRCN for horizons of 5 and 15 minutes and better for the intermediate case. The required time of the LSTM is between 22 and 24 seconds for the two shortest predictions horizons and around 7 seconds for the largest. On the other hand, the GRU model shows the worst times, although the times are not so big (between 40 and 52 seconds). GRU time decreases when the prediction horizon grows.

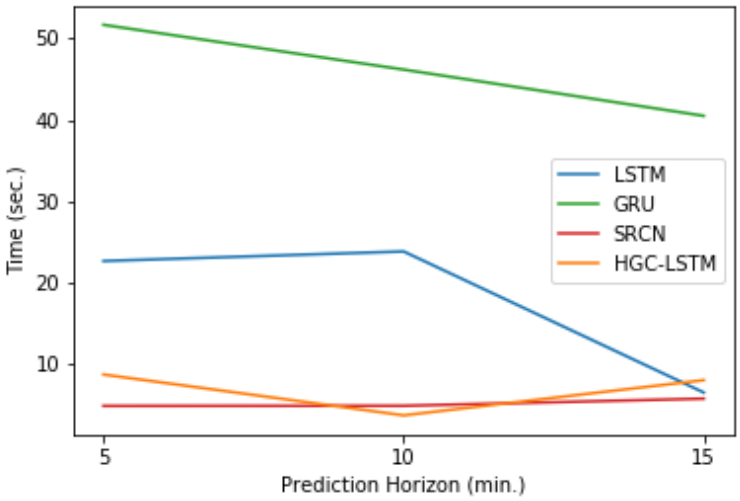


Figure 5.13: Plots of the training computational time of the models in function of the short-term prediction horizon in the Camp Nou network.

Camp Nou network results for long-term prediction horizon

This Subsection exposes the results of the proposed experiments executed over the Camp Nou network (described in Section 5.1.2). These results are shown in Table 5.8. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.14 shows a plot for each error measure. These plots contain the error for each model and for each long-term prediction horizon.

Compared with the short-term results, the errors of the convolutional methods improves for long-term predictions and worse for the GRU and LSTM options. Having said that, like in the short-term experiments, GRU is the method that shows best results for the tested horizons in Camp Nou network.

Experiment	**MAE (km/h)**	**RMSE (km/h)**	**Training time (Sec.)**
EX-LSTM-CN-PH20	2.991	4.914	24.6
EX-LSTM-CN-PH40	2.953	4.863	23.3
EX-LSTM-CN-PH60	3.063	4.923	18.1
EX-GRU-CN-PH20	2.913	4.804	39.6
EX-GRU-CN-PH40	2.893	4.803	47.0
EX-GRU-CN-PH60	2.927	4.874	45.9
EX-SRCN-CN-PH20	3.168	5.083	5.1
EX-SRCN-CN-PH40	2.945	4.865	17.0
EX-SRCN-CN-PH60	3.021	4.935	18.0
EX-HGC-CN-PH20	3.165	5.084	4.0
EX-HGC-CN-PH40	3.201	5.077	13.7
EX-HGC-CN-PH60	3.186	5.062	8.9

Table 5.8: Results of the long-term prediction horizon experiments in the Camp Nou network.

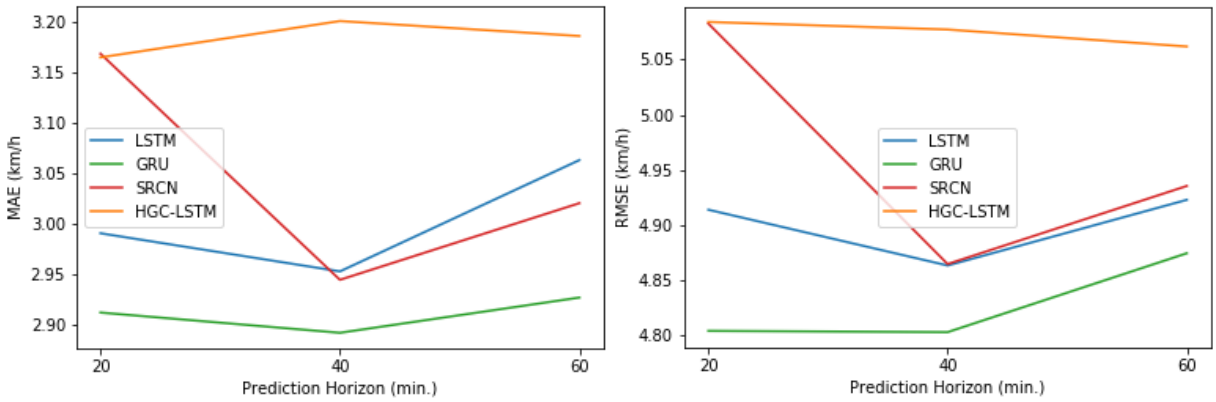


Figure 5.14: Plots of the MAE (left) and RMSE (right) errors of the models in function of the long-term prediction horizon in the Camp Nou network.

In the opposite side, the error in the HGC-LSTM predictions is, in general, the highest. The SRCN method presents similar results than the HGC-LSTM for the shortest horizon (20 minutes), but it strongly improves for the other tests, being the second best forecasting method for the other long horizons. Lastly, the LSTM option shows the second lowest error for the predictions of the 20 minutes horizon. It improves its predictions for the second temporal horizon, although it is outperformed by the SRCN method. But, for the last test, its error increase until its worst result. Except for the HGC-LSTM method, the test of the 40 minutes horizon shows a generalized improvement in all the methods.

As it is shown in Figure 5.15, with the long-term prediction horizons, the training time of the SRCN and HGC-LSTM methods increases. Contrary, in general, the time of the LSTM and GRU options is similar than the previous experiments. Despite this, GRU remains to be the slowest method and the convolutional ones the fastest. LSTM shows intermediate results and a smooth improvement for the longest horizon, matching the learning time of the SRCN method.

Amara network results for short-term prediction horizon

This Subsection exposes the results of the proposed experiments executed over the Amara network (described in Section 5.1.2). These results are shown in Table 5.9. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.16 shows a plot for each error measure. These plots contain the error for each model and for each short-term prediction horizon.

The error results for this experiment shows an interesting behavior in the two kinds of tested methods. LSTM and GRU perform very accurate predictions for the three prediction horizons despite their error grow gradually when the horizon increases. SRCN and HGC-LSTM show a strong descending error as the prediction horizon increases. In fact, the error for all the methods is very similar for largest horizon (15 minutes), with a slight advantage for the convolutional methods.

In terms of computational learning time, the results of Figure 5.17 shows a different trend between

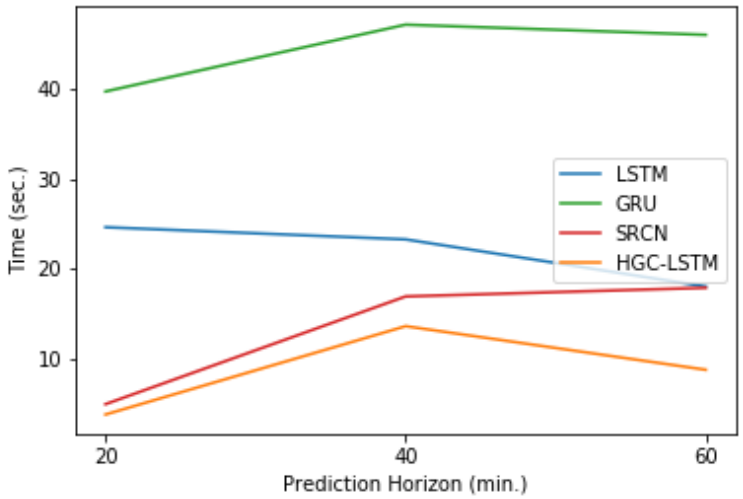


Figure 5.15: Plots of the training computational time of the models in function of the long-term prediction horizon in the Camp Nou network.

Experiment	**MAE (km/h)**	**RMSE (km/h)**	**Training time (Sec.)**
EX-LSTM-AM-PH5	1.831	4.602	5.8
EX-LSTM-AM-PH10	1.9	4.748	11.9
EX-LSTM-AM-PH15	1.929	4.835	12.4
EX-GRU-AM-PH5	1.88	4.73	21.5
EX-GRU-AM-PH10	1.861	4.699	23.0
EX-GRU-AM-PH15	1.898	4.794	22.5
EX-SRCN-AM-PH5	3.364	5.892	5.8
EX-SRCN-AM-PH10	2.477	5.076	7.5
EX-SRCN-AM-PH15	1.852	4.792	12.2
EX-HGC-AM-PH5	2.905	5.412	5.5
EX-HGC-AM-PH10	3.317	5.84	4.9
EX-HGC-AM-PH15	1.836	4.782	8.3

Table 5.9: Results of the short-term prediction horizon experiments in the Amara network.

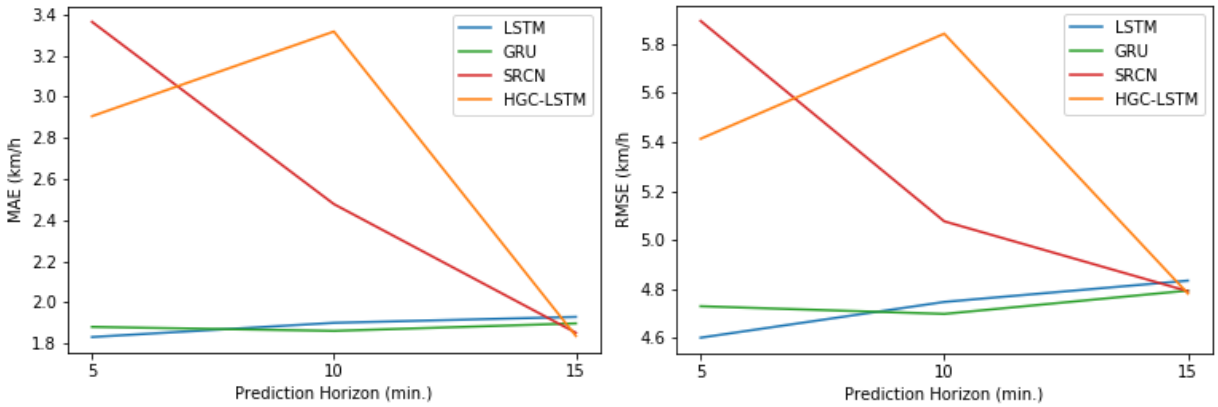


Figure 5.16: Plots of the MAE (left) and RMSE (right) errors of the models in function of the short-term prediction horizon in the Amara network.

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-AM-PH20	1.881	4.752	6.5
EX-LSTM-AM-PH40	1.879	4.795	5.3
EX-LSTM-AM-PH60	1.944	5.069	13.9
EX-GRU-AM-PH20	1.89	4.722	12.6
EX-GRU-AM-PH40	1.857	4.833	3.4
EX-GRU-AM-PH60	1.934	4.948	13.6
EX-SRCN-AM-PH20	1.85	4.811	10.9
EX-SRCN-AM-PH40	1.849	4.845	7.4
EX-SRCN-AM-PH60	1.838	4.86	6.5
EX-HGC-AM-PH20	1.845	4.807	9.5
EX-HGC-AM-PH40	3.349	5.914	5.4
EX-HGC-AM-PH60	1.851	4.861	9.5

Table 5.10: Results of the long-term prediction horizon experiments in the Amara network.

the GRU methods and the rest. While GRU has a constant time of around 22 seconds, the other methods have an incremental time as the prediction horizon increases. The maximum time of these three methods is of approximately 12 seconds for LSTM and SRCN, and 8 seconds in the HGC-LSTM case.

Amara network results for long-term prediction horizon

In this Subsection, the results of the proposed experiments executed over the Amara network (described in Section 5.1.2) are analyzed. These results are shown in Table 5.10. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.18 shows a plot for each error measure. These plots contain the error for each model and for each long-term prediction horizon.

For the long-term experiments over the Amara network, the four methods show similar errors. Although the error of the GRU and LSTM continue growing with the horizon, the MAE of all methods is between 1.8 and 1.95. Excepting the second test of the HGC-LSTM method, which is out of these bounds.

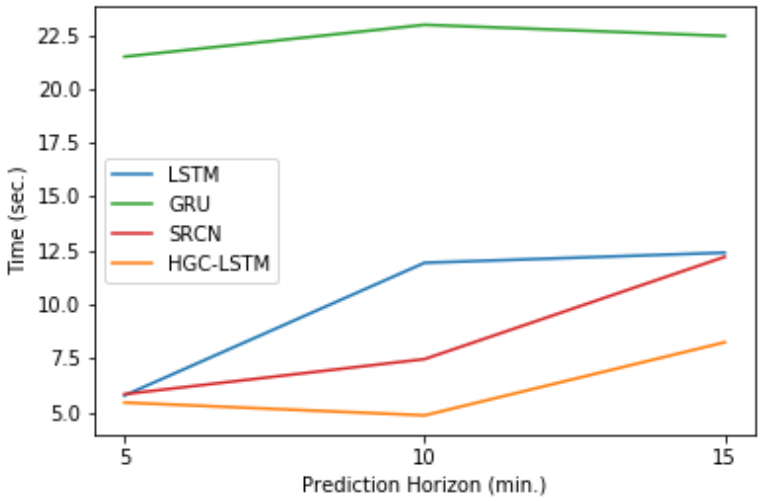


Figure 5.17: Plots of the training computational time of the models in function of the short-term prediction horizon in the Amara network.

A striking phenomenon is detected in this case whit an MAE of 3.5.

The computational training time of the four models it is very unstable, as can be seen in Figure 5.19. Consequently, for each tested prediction horizon, the fastest method is different. GRU and HGC-LSTM methods show the same pattern in which the times for the two extreme horizons (20 and 60 minutes) and the time of the intermediate case is very smaller. The LSTM option presents worse times when the horizon increases. Lastly, SRCN follows the opposite trend, it improves its learning time when the horizon grows.

Experiment conclusions

Lastly, the previously presented results are summarized and some conclusions are exposed. In general, the GRU method outperforms the others methods for all the performed experiments, for both Camp Nou and Amara networks. LSTM also shows good predictions for almost all the cases. In contrast, the convolutional methods are the worst options for all the cases except for the long-term predictions in the Amara scenario, where all methods perform with similar accuracy.

The accuracy of GRU and LSTM models is very similar for long-term and short-term experiments in the two scenarios. This shows that the prediction horizon is not so critical for these methods. The behavior of the convolutional methods changes depending on the network size. For the Camp Nou, the HGC-LSTM method performs better for the shortest horizons, while SRCN improves the accuracy with the longest prediction horizons. In the Amara case, both perform better with longer horizons, matching with the GRU and LSTM accuracy for the longest ones.

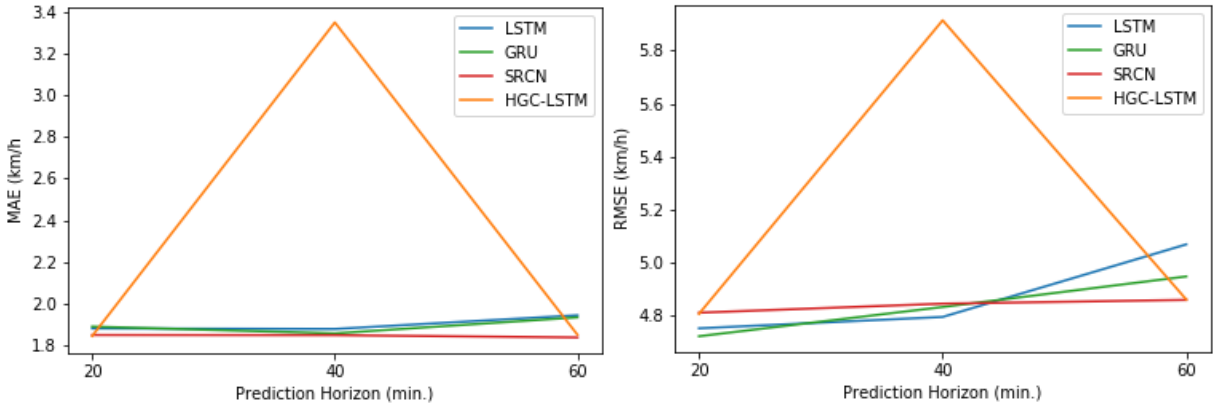


Figure 5.18: Plots of the MAE (left) and RMSE (right) errors of the models in function of the long-term prediction horizon in the Amara network.

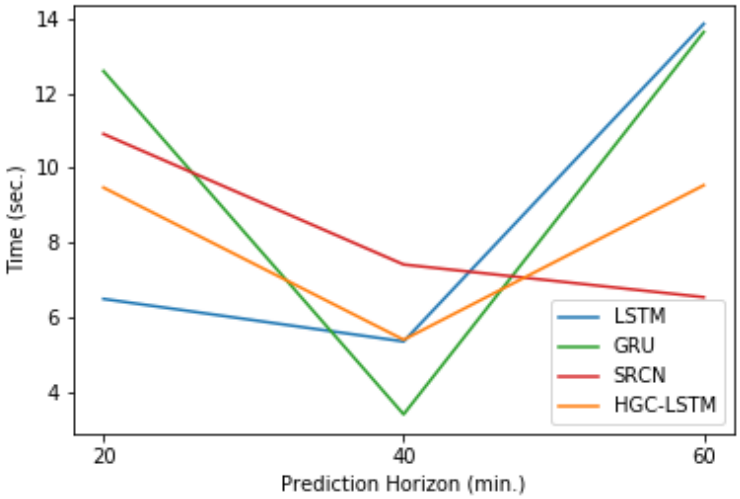


Figure 5.19: Plots of the training computational time of the models in function of the long-term prediction horizon in the Amara network.

In addition, the size is very relevant for the accuracy of the methods. Like in the penetration rate experiments, the developed models show smaller errors for the Amara scenario than for the Camp Nou one. And this does not depend on the prediction horizon, the improvement in the Amara network is generalized for all the methods and for almost all the tested horizons.

Moving on, the training computational time of the methods is also analyzed. The methods that need less time to learn the data are the HGC-LSTM and the SRCN, with a slight advantage for the first one. LSTM is the third option in this training time ranking and, lastly, the GRU method that requires more time.

In general terms, the prediction horizon does not affect the requires learning time. Only GRU and HGC-LSTM options show time improvements with the long-term predictions in the Amara network and, also the SRCN method works faster for short-term horizons in the Camp Nou scenario.

Finally, the effect of the network size to the learning time is considered. Overall, the methods learn faster with the big scenario. As an exception, the SRCN model performs the training for the short-term predictions in less time than for the Amara scenario.

5.4.3 Amount of training data

The amount of historical data needed to train the methods is an important consideration for the traffic forecasting applications. In particular, it is quantified by the number of days corresponding to the data used. Typically, the more available data, a better machine learning models performing and generalization is achieved, but it also presents some disadvantages. The first one is that the data gathering has an associated economic cost, and this increases with the amount of data. Another is the possible changes in the real network or in users behavior. So, the use of old data, which is a consequence of the increment in the amount of data, can train the model with old information which could harm future predictions or simply be useless.

The following experiments evaluate the performance of each model with different amount of training data. Also, the importance of the network size is studied in order to determine what models need more data to perform accurate predictions for each situation.

Experiment design

Table 5.11 summarizes the proposed experiments and details the corresponding design factor combinations. The experiments combine the four different prediction models, the two different amounts of training data (5, 10 and 15) in days and the two proposed scenarios (Camp Nou and Amara).

The different amounts of data selected are conditioned by some factors. The used simulator does not allow to perform simulations longer than 20 days. From these available days, two of them are reserved for methods testing purposes, so the maximum training data available corresponds to 18 days. Inside of this

Experiment	**Model**	**Scenario**	**Training data (days)**
EX-LSTM-CN-TD5	LSTM	Camp Nou	5
EX-LSTM-CN-TD10			10
EX-LSTM-CN-TD15			15
EX-LSTM-AM-TD5		Amara	5
EX-LSTM-AM-TD10			10
EX-LSTM-AM-TD15			15
EX-GRU-CN-TD5	GRU	Camp Nou	5
EX-GRU-CN-TD10			10
EX-GRU-CN-TD15			15
EX-GRU-AM-TD5		Amara	5
EX-GRU-AM-TD10			10
EX-GRU-AM-TD15			15
EX-SRCN-CN-TD5	SRCN	Camp Nou	5
EX-SRCN-CN-TD10			10
EX-SRCN-CN-TD15			15
EX-SRCN-AM-TD5		Amara	5
EX-SRCN-AM-TD10			10
EX-SRCN-AM-TD15			15
EX-HGC-CN-TD5	HGC-LSTM	Camp Nou	5
EX-HGC-CN-TD10			10
EX-HGC-CN-TD15			15
EX-HGC-AM-TD5		Amara	5
EX-HGC-AM-TD10			10
EX-HGC-AM-TD15			15

Table 5.11: Set of the proposed experiments for the amount of training data.

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-CN-TD5	2.931	4.865	22.8
EX-LSTM-CN-TD10	2.853	4.645	16.3
EX-LSTM-CN-TD15	2.691	4.607	77.7
EX-GRU-CN-TD5	2.862	4.776	49.6
EX-GRU-CN-TD10	2.719	4.493	91.9
EX-GRU-CN-TD15	2.749	4.629	77.3
EX-SRCN-CN-TD5	3.041	4.91	8.0
EX-SRCN-CN-TD10	3.037	4.947	5.8
EX-SRCN-CN-TD15	2.765	4.662	45.0
EX-HGC-CN-TD5	3.103	5.03	8.5
EX-HGC-CN-TD10	2.89	4.676	24.6
EX-HGC-CN-TD15	2.784	4.69	23.8

Table 5.12: Results of the amount of training data experiments in the Camp Nou network.

training dataset, a percentage of data is used for the validation of the method. In the rest of the proposed experiments, the data used corresponds to 7 days: 5 days for training, with 1 for validation (20%), and 2 for testing. In order to do not change the percentage of validation data and to avoid use incomplete days for this, the only available configurations are the use of 5, 10 and 15 days for training.

Camp Nou network results

This Subsection exposes the results of the proposed experiments executed over the Camp Nou network (described in Section 5.1.2). These results are shown in Table 5.12. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.20 shows a plot for each error measure. These plots contain the error for each model and for each amount of training data.

As it is expected, the error of almost all the models decreases each time the amount of data used for training increases. In this scenario, the GRU and LSTM methods performs better than the rest for all the experiments. In particular, GRU is able to predict with more accuracy than the LSTM for the first two cases and this trend change with 15 days of data. Regarding to the convolutional methods, SRCN shows an slightly lower error than HGC-LSTM for 5 and 15 days experiments. In contrast, HGC-LSTM performs more accurate predictions than SRCN for the intermediate case.

In terms of computational learning time, the results of Figure 5.21 show a generally incremental trend with more data. In particular, the SRCN and LSTM models show a slight improvement between the 5 and the 10 days cases, with a strong time increment for the last experiment (15 days). In contrast, the GRU and HGC-LSTM methods present a strong growth between the two experiments with less data, but their time decreases for the 15 days case. SRCN and HGC-LSTM proposals are the fastest for almost all the experiments. Contrary, GRU is the slowest method for all cases. LSTM is between the two groups, except for the intermediate test, where it trains faster than the HGC-LSTM method.

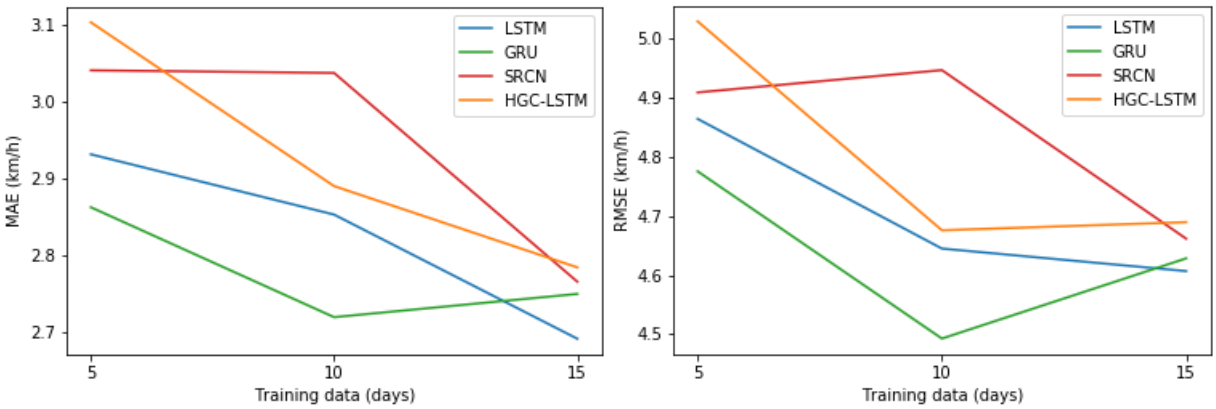


Figure 5.20: Plots of the MAE (left) and RMSE (right) errors of the models in function of the amount of training data in the Camp Nou network.

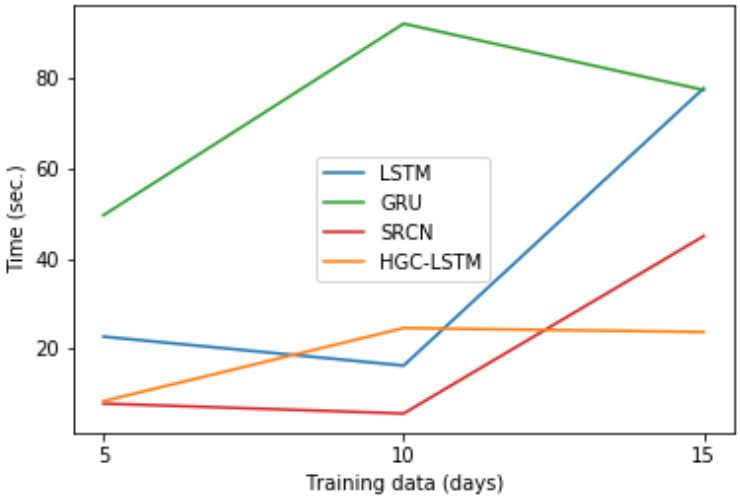


Figure 5.21: Plots of the training computational time of the models in function of the amount of training data in the Camp Nou network.

Experiment	MAE (km/h)	RMSE (km/h)	Training time (Sec.)
EX-LSTM-AM-TD5	1.812	4.097	12.3
EX-LSTM-AM-TD10	1.412	3.085	5.0
EX-LSTM-AM-TD15	1.566	3.551	29.8
EX-GRU-AM-TD5	1.8	4.057	21.1
EX-GRU-AM-TD10	1.608	3.537	13.2
EX-GRU-AM-TD15	1.567	3.524	44.2
EX-SRCN-AM-TD5	1.581	3.509	12.5
EX-SRCN-AM-TD10	2.545	4.159	7.4
EX-SRCN-AM-TD15	1.466	3.571	36.8
EX-HGC-AM-TD5	1.581	3.509	12.5
EX-HGC-AM-TD10	3.402	5.349	6.5
EX-HGC-AM-TD15	1.47	3.579	34.1

Table 5.13: Results of the amount of training data experiments in the Amara network.

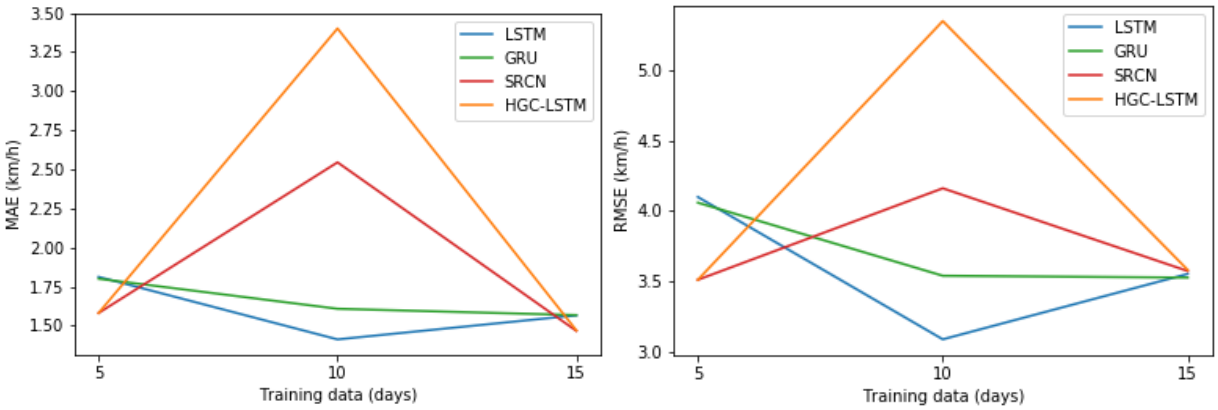


Figure 5.22: Plots of the MAE (left) and RMSE (right) errors of the models in function of the amount of training data in the Amara network.

Amara network results

In this Subsection, the results of the proposed experiments executed over the Amara network (described in Section 5.1.2) are analyzed. These results are shown in Table 5.13. To begin with, the results for the MAE and the RMSE measures are presented. Figure 5.22 shows a plot for each error measure. These plots contain the error for each model and for each proposed amount of training data.

The error of the methods in function to the amount of training data shows a different trend for the methods with convolutional components and the others. While the first ones have smaller errors for the 5 and 15 days cases than for 10 days, the results of the GRU and LSTM methods shows a better accuracy for the 10 days experiment than for the others. When the models are trained with 5 days corresponding data, SRCN and HGC-LSTM have practically the same error and they are the best options with a slight advantage in front of the rest. In the 10 days, LSTM and GRU methods strongly improve the error of the convolutional ones, in particular, LSTM performs better. Lastly, for the 15 days experiment, all the methods have a very similar error.

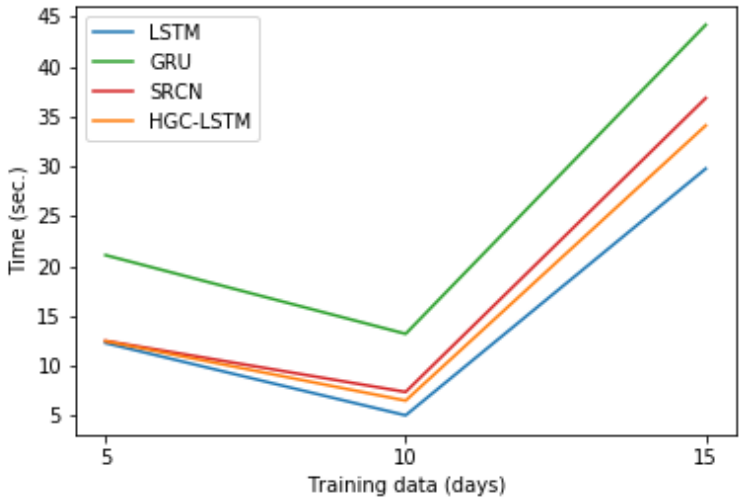


Figure 5.23: Plots of the training computational time of the models in function of the amount of training data in the Amara network.

As it is shown in Figure 5.23, all the methods follow a similar training time trend, but with different values. They decrease the learning time from the first experiment to the second one and strongly increases it for the 15 days test. The three fastest options are LSTM, HGC-LSTM, and SRCN in this order, and the difference between them increases with more data. GRU is slower than the rest with a bigger difference for all the experiments.

Experiment conclusions

The results presented above show some interesting conclusions of the performed experiments. The methods that perform better, in general, are the LSTM and GRU. In particular, GRU shows a smaller error for the Camp Nou network and LSTM is the best option for the Amara scenario. The convolutional solutions are better in a twice of cases for the Amara network, but the difference is small.

The accuracy of the proposed forecasting methods generally increases with more data, but the improvement is not so big. This could be related to the recurrent pattern used for the data generation (Section 5.1). Because the same pattern repeats itself, the extra data does not provide new information to the models. Figure 5.24 shows how the average speed of the network sections evolves during the time for the Camp Nou scenario. Thus, although the methods perform better with more data, the difference is not critical, especially in the Amara network, where the improvement is minimal.

In addition, the developed traffic forecasting methods are evaluated in terms of training time. Here, the results are similar to the other experiments. The SRCN and HGC methods are the most stable in the learning speed. GRU is the slowest method for all the performed experiment, independently to the amount of data and the network. Although the previous facts, LSTM is the fastest option for the Amara

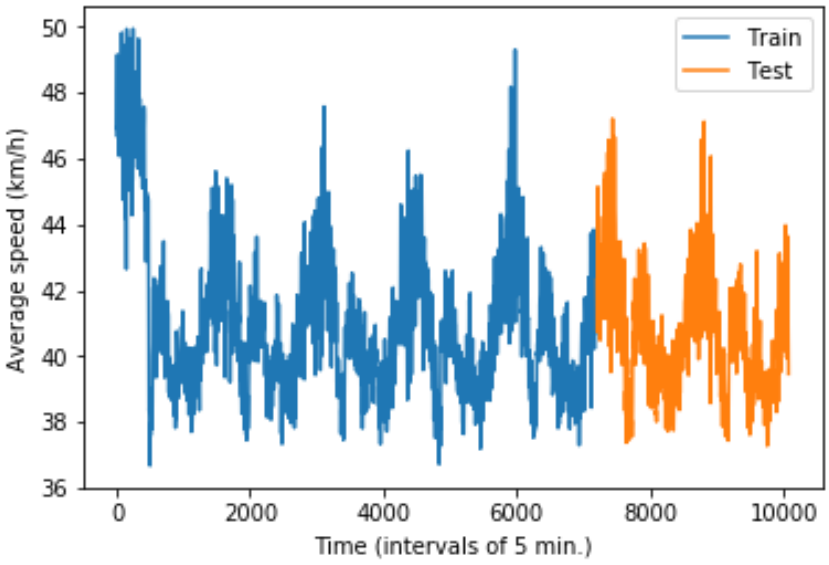


Figure 5.24: Plot of the average speed of all the network sections of the Camp Nou scenario during 7 days of data.

scenario but is not so good for the Camp Nou one. The size of the network affects strongly to the GRU and LSTM, whose shows better times for the Amara network. But it is not so relevant for the methods with a convolutional component.

Chapter 6

Final comments and conclusions

This final chapter looks back on the main achievements of this master thesis. The summary of the proposed objectives and the obtained results are presented. After this, the research contributions are discussed. Finally, some recommendation for further research are formulated.

6.1 Conclusions

This master thesis deals with the traffic forecasting problem. As it is shown in the state of the art (Section 2), the research in this topic has been very active in the last 40 years. This is because traffic forecasting plays a key role in mitigating some traffic and transportation problems. In particular, this project is focused on traffic forecasting in urban networks using Floating Car Data (FCD). The objectives and the research questions described in Section 1.3 are revisited and some conclusions regarding them are presented.

Thus, the main objective of the work presented in this master thesis is to perform traffic forecasting in urban contexts when FCD is available using different machine learning solutions. In the project, an extensive state of the art is done in order to study which methods are proposed in the literature. On this basis, four deep learning methods are developed and analyzed in order to solve this problem. They are validated with different experiments designed through a microscopic traffic simulation approach to emulate FCD. As it is more concretely commented in the conclusions of the performed experiments (Section 5.4) and in the following paragraphs, these methods present very good performance.

The first research question presented in this project suggests to analyze how the penetration rate of FCD affects the traffic forecasting performance. In order to offer an answer to this question, the experiments of Subsection 5.4.1 are proposed. The results of these experiments show that, as it is expected, the accuracy of the predictions increases when the penetration rate grows. Having said this, the evolution of the error according to the penetration rate is different for the two tested networks. The smallest scenario

(Camp Nou) presents a stronger increment of the error when the number of vehicles decreases while the biggest one (Amara) shows a smoother evolution. Although this, the errors of the developed methods for the worst tested situation (25% of floating cars) are reasonably low for the problem. Anyhow, the performing of additional experiments testing the methods with lower penetration rate could reveal the real limits of these solutions.

The second and third questions require a comparison between the developed forecasting traffic methods. From the performed computational experiences, the outperformance of the GRU and LSTM models can be observed for almost all situations. Depending on the case, one of them outperforms the other and vice versa, but the difference between their errors measurements is usually small. Having said this, in terms of required training time, the convolutional methods show a faster learning process for the most tested cases. In these terms, GRU is generally the slowest developed forecasting method. In relation to the use of the topological information by the SRCN and HGS-LSTM, these methods do not show evidence of a significant improvement. They show better results for some tested situations, but with small accuracy differences.

In addition, all the experiments are performed in two urban scenarios of different sizes. Although the smallest (Camp Nou) is bigger than the commonly used in other research works, the methods are also tested with a scenario equivalent to a city district (Amara). The main trend of the research findings of this master thesis shows that the methods perform better predictions, and require less training time for the Amara network. This proves that the developed methods can perform accurate forecasting in big urban zones, although it could be interesting to test them in the network of a whole city.

One of the main issues in the traffic forecasting problem is the prediction horizon, which can be short-term or long-term. The included computational experiences study how every method works in both terms. In general, LSTM and GRU prove that the prediction horizon is not so critical for them because their accuracy is very similar for long-term and short-term experiments in the two scenarios. This is not the case of the convolutional methods. These perform better with the longest prediction horizons in the Amara scenario. In the Camp Nou network, HGC-LSTM predicts traffic better for the shortest horizons, while SRCN shows better results with the longest ones.

Finally, an important consideration for the traffic forecasting application is the amount of historical data needed to train the methods. Increasing the historical data could improve the predictions, but it has an economic cost and the changes in the real network are more likely. The accuracy of the proposed forecasting methods generally increases with more data, but the difference is not critical, especially in the Amara network, where the improvement is minimal. So, the use of approximately 5 days of data is enough to perform good predictions in a recurrent situation, like the one tested.

6.2 Contributions

In addition to the achievement of the objectives and research questions proposed in this master thesis, it also presents its contributions whose are described in this Section.

The first contribution is the development and testing of the SRCN and HGC-LSTM methods using GRU layers as an alternative to LSTM typically used. As it is shown in the state of the art (Section 2), Fu et al. [2017] conclude that GRU outperforms LSTM in traffic forecasting. Thus, if the GRU method is able to improve the LSTM results individually, then it could improve the accuracy of the SRCN and HGC-LSTM methods by replacing the LSTM layers by GRU layers.

Although the SRCN and HGC-LSTM methods are compared in the literature with the LSTM method separately, these ones are not compared with GRU model nor between them. So, this project performs an intensive comparison between the four deep learning solutions with the same conditions. This allows to evaluate their capabilities in different situations and contrast which one predicts the traffic more precisely.

Another contribution is the development of a simulation-based framework to test different traffic forecasting methods with different scenarios and custom situations. This tool can be used to propose new experiments by adding new methods, networks and data sources, and configuring multiple traffic forecasting problem issues.

Finally, part of the work of this master thesis is included in an extended abstract, named *A Comparison of Deep Learning Methods for Urban Traffic Forecasting using Floating Car Data. J.J. Vázquez, J. Arjona, M.P. Linares, J. Casanovas-Garcia*, submitted to the *22nd Meeting of the Euro Working Group on Transportation (EWGT2019)*.

6.3 Further Research

Although the proposed goals and the exposed contributions have been achieved, this project opens some further research lines to extend the obtained results. This Section exposes some of the most interesting ones.

First of all, the performing of more experiments for the proposed factors could be revealing to refine the conclusions presented. For example, the testing of lower penetration rates or different amounts of training data. Besides the extension of the existing experiments, the performing of new ones with factors such as the time windows or the frequency of the data collection. Also, the convolutional methods have shown worse results than the rest and could be interesting to test them with different input configurations (the parameter K for the HGC-LSTM and the map size for the SRCN).

In almost all of the presented experiments, particularly in the Amara network, the errors of the RMSE are almost twice the MAE. This can be interpreted as proof that the traffic forecasting error is more strong in some sections of the network. Thus, future work could be the analysis of the error distribution

over the different network sections and reveal the more traffic conflicting zones. In addition to the spatial analysis of the error distribution, the temporal distribution can also help to detect the hours of the day when the traffic prediction is more difficult.

As it is shown in Section 5.1, the FCD generated in the project follows a recurrent pattern for every day. This can allow the deep learning models to learn this pattern and perform accurate predictions based exclusively on historical data. To check if this is happening, a good future research point is the performing of more realistic simulations. The inclusion of different patterns for different days or the random generation of special situations like events or traffic accidents could be good future issues.

In a similar line to the previous proposal, another future issue is the testing of the developed methods using real FCD. It is clear that the performing of some experiments it is not possible using real data because some situations, like having penetration rate of 100%, are currently a utopia in a real urban network. But the simulation of a scenario for which the real data is available could help to compare how the simulation results fit with reality.

As in most data-driven projects, the presence of missing values it is one of the most frequent and tedious problems. In this case, the FCD can be insufficient to cover all the network sections in all periods, which results in missing values. The solution most widely used is the imputation of these values from the rest of historical data, but it is not possible when a variable (a section in this problem) has no values for any period. In this project, these cases have been filled with the maximum allowed speed of the sections, assuming that the sections without data are empty sections practically always. The research in new approaches to face the missing values problem is a good future extension of the project.

A possible solution for the missing values problem can be the addition of new data sources. Besides this problem, the inclusion of new data sources can complement the FCD and improve the forecasting accuracy. The new data could be of the same type of the one already used (e.g. speeds from loop sensors) or completely different (exogenous variables like weather conditions or calendar events). Many cities have already installed sensors to measure, not only the traffic but also different day-to-day aspects in their environment. And it is interesting to show how the information from these sensors can be used in traffic forecasting too.

Although in the literature forecasting models are usually tested with smaller urban networks than Amara, the use of a bigger network could be interesting. The study of the forecasting solutions proposed in a more complex scenario can confirm if they can perform accurate predictions with a reasonable training computational time and so if they are a feasible solution for big cities.

Another issue to perform more realistic predictions is the differentiation between the section lanes. In some real cases, the congestion of the sections is not homogeneous between their lanes. This is understandable since frequently, the lanes of a road do not have the same properties nor connections with other sections. So, each lane presents different situations although they belong to the same section.

The inclusion of new methods is always a possible extension for a project like this. The research activity in this field is very active and new solutions are proposed constantly. As it is shown in Chapter 2, a lot

of authors write about solutions with a different approach to the Deep Learning ones. The comparison of solutions with different approaches can be determinant to evaluate if the traffic forecasting results depend on the approach used, and if exists one which is the most appropriate for this problem. Also, the addition of some simple forecasting as a baseline method could help to evaluate how the proposed methods improve some basic solutions.

Bibliography

Abadi, Afshin, Tooraj Rajabioun, and Petros A. Ioannou (2015). "Traffic Flow Prediction for Road Transportation Networks With Limited Traffic Data". In: *IEEE Transactions on Intelligent Transportation Systems*. ISSN: 15249050. DOI: 10.1109/TITS.2014.2337238.

Bergstra, J. and B. Yoshua (2012). "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research*. ISSN: 1532-4435. DOI: 10.1162/153244303322533223. arXiv: 1703.05423.

Bezuglov, Anton and Gurcan Comert (2016). "Short-term freeway traffic parameter prediction: Application of grey system theory models". In: *Expert Systems with Applications*. ISSN: 09574174. DOI: 10.1016/j.eswa.2016.06.032.

Cai, Pinlong et al. (2016). "A spatiotemporal correlative k-nearest neighbor model for short-term traffic multistep forecasting". In: *Transportation Research Part C: Emerging Technologies*. ISSN: 0968090X. DOI: 10.1016/j.trc.2015.11.002. arXiv: 1512.08580.

Chen, Chen et al. (2017). *A short-term traffic prediction model in the vehicular cyber-physical systems*. DOI: 10.1016/j.future.2017.06.006.

Cheng, Shifen et al. (2018). "Short-term traffic forecasting: An adaptive ST-KNN model that considers spatial heterogeneity". In: *Computers, Environment and Urban Systems*. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2018.05.009.

Cheng, Xingyi et al. (2017). "DeepTransport: Learning Spatial-Temporal Dependency for Traffic Condition Forecasting". In: *CoRR* abs/1709.0. arXiv: 1709.09585. URL: http://arxiv.org/abs/1709.09585.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Cui, Zhiyong et al. (2018). "High-Order Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting". In: arXiv: 1802.07007. URL: http://arxiv.org/abs/1802.07007.

Du, Xunsheng et al. (2018). "Stacked LSTM deep learning model for traffic prediction in vehicle-to-vehicle communication". In: *IEEE Vehicular Technology Conference*. ISBN: 9781509059355. DOI: 10.1109/VTCFall.2017.8288312.

Fu, Rui, Zuo Zhang, and Li Li (2017). "Using LSTM and GRU neural network methods for traffic flow prediction". In: *Proceedings - 2016 31st Youth Academic Annual Conference of Chinese Association of Automation, YAC 2016*. ISBN: 9781509044238. DOI: 10.1109/YAC.2016.7804912.

Fusco, Gaetano et al. (2015). "Short-term traffic predictions on large urban traffic networks: Applications of network-based machine learning models and dynamic traffic assignment models". In: *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*. ISBN: 978-9-6331-3140-4. DOI: 10.1109/MTITS.2015.7223242.

Guo, Fangce, John W. Polak, and Rajesh Krishnan (2018). "Predictor fusion for short-term traffic forecasting". In: *Transportation Research Part C: Emerging Technologies*. ISSN: 0968090X. DOI: 10.1016/j.trc.2018.04.025.

Haworth, James and Tao Cheng (2012). "Non-parametric regression for space-time forecasting under missing data". In: *Computers, Environment and Urban Systems*. ISSN: 01989715. DOI: 10.1016/j.compenvurbsys.2012.08.005.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation*. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944.

Khan, Ata M. (2012). "Bayesian predictive travel time methodology for advanced traveller information system". In: *Journal of Advanced Transportation*. ISSN: 01976729. DOI: 10.1002/atr.147.

Lana, Ibai et al. (2018). "Road Traffic Forecasting: Recent Advances and New Challenges". In: *IEEE Intelligent Transportation Systems Magazine* 10.2, pp. 93–109. ISSN: 1939-1390. DOI: 10.1109/MITS.2018.2806634. URL: https://ieeexplore.ieee.org/document/8344781/.

Li, Yunxuan et al. (2017). "Taxi booking mobile app order demand prediction based on short-term traffic forecasting". In: *Transportation Research Record: Journal of the Transportation Research Board*. ISSN: 03611981. DOI: 10.3141/2634-10.

Lint, Hans van and Chris van Hinsbergen (2012). "Short-Term Traffic and Travel Time Prediction Models". In: *Artificial Intelligence Applications to Critical Transportation Issues*. Washington, D.C.: Transportation Research Board. Chap. Artificial, pp. 22–41. ISBN: 978-0-309-43481-2. DOI: 10.17226/22690. URL: https://www.nap.edu/catalog/22690.

Liu, Y. et al. (2017). "Short-term travel time prediction by deep learning: A comparison of different LSTM-DNN models". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. DOI: 10.1109/ITSC.2017.8317886.

Ma, Xiaolei et al. (2017). "Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction". In: *Sensors (Switzerland)*. ISSN: 14248220. DOI: 10.3390/s17040818. arXiv: 1701.04245.

Mendes-Moreira, João et al. (2015). "Improving the accuracy of long-term travel time prediction using heterogeneous ensembles". In: *Neurocomputing*. ISSN: 18728286. DOI: 10.1016/j.neucom.2014.08.072.

Moreira-Matias, Luís et al. (2016). "An online learning approach to eliminate Bus Bunching in real-time". In: *Applied Soft Computing Journal*. ISSN: 15684946. DOI: 10.1016/j.asoc.2016.06.031.

Pan, Yuanyuan and Yongdong Shi (2017). "A Grey Neural Network Model Optimized by Fruit Fly Optimization Algorithm for Short-term Traffic Forecasting." In: *engineeringletters.com*.

Polson, Nicholas G. and Vadim O. Sokolov (2017). "Deep learning for short-term traffic flow prediction". In: *Transportation Research Part C: Emerging Technologies*. ISSN: 0968090X. DOI: 10.1016/j.trc.2017.02.024. arXiv: 1604.04527.

Raza, Asif and Ming Zhong (2017). "Hybrid lane-based short-term urban traffic speed forecasting: A genetic approach". In: *2017 4th International Conference on Transportation Information and Safety, ICTIS 2017 - Proceedings*. ISBN: 9781538604373. DOI: 10.1109/ICTIS.2017.8047776.

Salamanis, Athanasios et al. (2016). "Managing Spatial Graph Dependencies in Large Volumes of Traffic Data for Travel-Time Prediction". In: *IEEE Transactions on Intelligent Transportation Systems*. ISSN: 15249050. DOI: 10.1109/TITS.2015.2488593.

Scalabrin, Maria et al. (2017). "A Bayesian forecasting and anomaly detection framework for vehicular monitoring networks". In: *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*. ISBN: 9781509063413. DOI: 10.1109/MLSP.2017.8168151.

Schimbinschi, Florin et al. (2017). "Topology-regularized universal vector autoregression for traffic forecasting in large urban areas". In: *Expert Systems with Applications*. ISSN: 09574174. DOI: 10.1016/j.eswa.2017.04.015.

Sun, Bin et al. (2017). "An Overview of Parameter and Data Strategies for k-Nearest Neighbours Based Short-Term Traffic Prediction". In: *Proceedings of the 2017 International Conference on E-Society, E-Education and E-Technology - ICSET 2017*. New York, New York, USA: ACM Press, pp. 68–74. ISBN: 9781450353762. DOI: 10.1145/3157737.3157749. URL: http://dl.acm.org/citation.cfm?doid=3157737.3157749.

Sun, Shiliang and Changshui Zhang (2007). "The selective random subspace predictor for traffic flow forecasting". In: *IEEE Transactions on Intelligent Transportation Systems*. ISSN: 15249050. DOI: 10.1109/TITS.2006.888603.

Sun, Shiliang, Rongqing Huang, and Ya Gao (2012). "Network-Scale Traffic Modeling and Forecasting with Graphical Lasso and Neural Networks". In: *Journal of Transportation Engineering* 138.11, pp. 1358–1367. ISSN: 0733-947X. DOI: 10.1061/(ASCE)TE.1943-5436.0000435.

Vlahogianni, Eleni I., Matthew G. Karlaftis, and John C. Golias (2014). "Short-term traffic forecasting: Where we are and where we're going". In: *Transportation Research Part C: Emerging Technologies* 43, pp. 3–19. ISSN: 0968090X. DOI: 10.1016/j.trc.2014.01.005. URL: https://www.sciencedirect.com/science/article/pii/S0968090X14000096.

Wang, Jiaqiu, Ioannis Tsapakis, and Chen Zhong (2016). "A space-time delay neural network model for travel time prediction". In: *Engineering Applications of Artificial Intelligence*. ISSN: 09521976. DOI: 10.1016/j.engappai.2016.02.012.

Wibisono, Ari et al. (2016). "Traffic big data prediction and visualization using Fast Incremental Model Trees-Drift Detection (FIMT-DD)". In: *Knowledge-Based Systems*. ISSN: 09507051. DOI: 10.1016/j.knosys.2015.10.028.

Wu, Yao-Jan et al. (2016). "Urban Traffic Flow Prediction Using a Spatio-Temporal Random Effects Model". In: *Journal of Intelligent Transportation Systems*. ISSN: 1547-2450. DOI: 10.1080/15472450.2015.1072050.

Xia, Dawen et al. (2016). "A distributed spatial-temporal weighted model on MapReduce for short-term traffic flow forecasting". In: *Neurocomputing*. ISSN: 18728286. DOI: 10.1016/j.neucom.2015.12.013.

Xu, Jie et al. (2015). "Mining the Situation: Spatiotemporal Traffic Prediction with Big Data". In: *IEEE Journal on Selected Topics in Signal Processing*. ISSN: 19324553. DOI: 10.1109/JSTSP.2015.2389196.

Xu, Yanyan et al. (2016). "Urban traffic flow prediction: a spatio-temporal variable selection-based approach". In: *Journal of Advanced Transportation* 50.4, pp. 489–506. ISSN: 01976729. DOI: 10.1002/atr.1356. URL: http://doi.wiley.com/10.1002/atr.1356.

Yanjie Duan, Yisheng Lv, and Fei-Yue Wang (2016). "Travel time prediction with LSTM neural network". In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. ISBN: 978-1-5090-1889-5. DOI: 10.1109/ITSC.2016.7795686.

Ye, Qing, W. Y. Szeto, and S. C. Wong (2012). "Short-term traffic speed forecasting based on data recorded at irregular intervals". In: *IEEE Transactions on Intelligent Transportation Systems*. ISSN: 15249050. DOI: 10.1109/TITS.2012.2203122.

Yi, Hongsuk, Hee Jin Jung, and Sanghoon Bae (2017). "Deep Neural Networks for Traffic Flow Prediction". In: *2017 Ieee International Conference on Big Data and Smart Computing (Bigcomp)*. ISSN: 2375-933X. DOI: 10.1109/BIGCOMP.2017.7881687.

Yu, Bing, Haoteng Yin, and Zhanxing Zhu (2017a). "Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting". In: *arXiv*. arXiv: 1709.04875.

Yu, Haiyang et al. (2017b). "Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks". In: *Sensors (Switzerland)*. ISSN: 14248220. DOI: 10.3390/s17071501. arXiv: 1705.02699.

Zhang, Yanru and Ali Haghani (2015). "A gradient boosting method to improve travel time prediction". In: *Transportation Research Part C: Emerging Technologies*. ISSN: 0968090X. DOI: 10.1016/j.trc.2015.02.019.

Zheng, Fangfang and Henk Van Zuylen (2013). "Urban link travel time estimation based on sparse probe vehicle data". In: *Transportation Research Part C: Emerging Technologies*. ISSN: 0968090X. DOI: 10.1016/j.trc.2012.04.007.

Zhu, Zheng et al. (2016). "Short-term traffic flow prediction with linear conditional Gaussian Bayesian network". In: *Journal of Advanced Transportation*. ISSN: 20423195. DOI: 10.1002/atr.1392.